Universidad Carlos III de Madrid
Escuela Politécnica Superior

Ingeniería en Informática
Proyecto de Fin de Carrera

nfcTicketing: Aplicación para uso de tiques de transporte público

Autor: Irene Amador Román
Tutor: José María Sierra Cámara
Fecha: Mayo de 2013

«The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards - and even then I have my doubts»

Eugene Howard Spafford

«The only place where success comes before work is in the dictionary»

Donald Kendall

Resumen

La evolución en los últimos años de las tecnologías de comunicación y las nuevas funcionalidades que empezaron a incluir los dispositivos móviles desembocaron en la aparición de los llamados *smartphones*. Estos dispositivos incorporan tecnologías como el hace tiempo implantado *Bluetooth*, sistemas de posicionamiento global (GPS), conexión a Internet tanto a través de Wi-Fi como 3G o la más reciente tecnología de comunicación a corta distancia, el *Near Field Communication* (NFC), que provocan que se deje de pensar en estos dispositivos como simples teléfonos. La tecnología NFC se plantea como una de las alternativas más importantes para la realización de pagos electrónicos y la interacción con el consumidor.

Uno de los paradigmas que puede cambiar esta tecnología es la compra y uso de un billete de transporte público de manera que se agilice todo el proceso a través de la utilización de un *smartphone*. Este proyecto plantea un sistema de uso de billetes de transporte público a través del teléfono móvil. Se realizará un análisis de la tecnología a emplear así como de otras aplicaciones similares ya estén implantadas o hayan sido probadas a través de proyectos piloto. Se pensará en los distintos entornos en los que podría ser usado el sistema teniendo siempre en cuenta las medidas de seguridad necesarias que deben ser tomadas para evitar un uso fraudulento del mismo.

Palabras clave: Android, NFC, Near Field Communication, Google, RFID, App, transporte público, e-ticket.

Abstract

The evolution in last years of communication technologies and the new features included in mobile devices led to the emergence of so-called smartphones. These devices incorporate technologies such as the long-implemented Bluetooth, Global Positioning System (GPS), Internet access via both Wi-Fi and 3G or the most recent short distance communication technology, Near Field Communication (NFC), which cause people stop thinking about these devices as simple phones. NFC technology is proposed as one of the most important alternatives for conducting electronics payments and consumer interactions.

One of the paradigms that this technology can change is the purchase and use of a public transport ticket so as to expedite the whole process through the utilization of a smartphone. This project proposes a system using public transport tickets via mobile phone. An analysis about the technology to be used as well as similar applications that are already implemented or have been tested through pilot projects has been realized. Different enviroments in which the system could be used have been taken into account along with the necessary security measures to be taken in order to prevent further misuse of it.

Keywods: Android, NFC, Near Field Communication, Google, RFID, App, public transport, e-ticket, transportation.

Agradecimientos

A mis padres, por animarme desde el principio a meterme en todo este embrollo, por siempre confiar en mi y por apoyarme en todos y cada uno de los pasos que me han hecho llegar aquí.

A mi hermano, por aguantar todas mis «friquezas», por ayudarme a desconectar cuando más falta me hacía, y por compartir tantas cosas conmigo, especialmente una de mis grandes pasiones. Porque ambos sabemos que la vida en rojo y blanco es mucho más divertida.

A Lisa, por hacer mucho más llevaderas todas esas horas estudio y prácticas, y por tantas y tantas horas de incansable compañía.

A mi familia, por siempre estar ahí.

A Sergio, Laura y Gregory, por aguantar mis quejas sobre asignaturas y prácticas de las que no entendían nada y por ser capaces de sacarme una sonrisa siempre que lo necesitaba.

A mis compañeros de la carrera, en especial a Alberto, Dani, Fombe, Héctor, Ja, Laura, Omar, Quique, Torres y Xabi por ser grandes compañeros de fatigas y convertirse en grandes amigos, por aprender codo con codo y a base de equivocarnos, y por hacer que cada instante compartido en la universidad sea un valioso recuerdo.

A mis compañeros del laboratorio Evalues, por brindarme su ayuda siempre que la he pedido. Y cuando no lo he hecho también.

A mi tutor, José María Sierra, por ofrecerme la posibilidad de trabajar en el laboratorio, por todo el apoyo y la ayuda prestada a lo largo del proyecto y por hacer que le diera una segunda oportunidad a la seguridad.

A todos, muchas gracias
Irene

Índice general

Índice de figuras

Índice de tablas

Parte I

Introducción

1

Introducción

1.1. Motivación

Desde que en 1983 Motorola lanzara al mercado el que se considera el primer teléfono móvil del mundo (Dynatac 8000x) el sector de la telefonía móvil a evolucionado a gran velocidad suponiendo una estupenda herramienta de trabajo diaria que permite manejar una gran cantidad de información en tiempo real. Las funcionalidades que ofrecen los teléfonos móviles han crecido rápidamente. Los primeros dispositivos únicamente permitían realizar llamadas. Posteriormente se incluyó en los terminales el servicio de mensajes cortos (del inglés, *Short Message System* - SMS). En los años siguientes se incluyeron nuevas funciones como reproducción de música, correo electrónico, agenda electrónica, fotografía digital, grabación y reproducción de vídeo digital, videollamada, sistema de posicionamiento global (del inglés *Global Positioning System* - GPS), Bluetooth, acelerómetro, giroscopio, sensor de proximidad... No obstante, la verdadera revolución en la telefonía móvil llegó de la mano de Internet móvil.

Con la aparición de Internet móvil cambió el concepto de los teléfonos móviles tal y como se conocían hasta la fecha y empezaron a comercializarse los *smartphones* que conocemos en la actualidad. Estos nuevos dispositivos con su gran variedad de sistemas operativos (Android, iOS, Windows Phone, Symbian, RIM...), sus nuevas características y su mejor rendimiento abrieron la posibilidad de desarrollar todo tipo de aplicaciones móviles que aprovecharan esto y mejoraran la experiencia del usuario.

En los últimos años el mercado de las aplicaciones móviles ha crecido exponencialmente estimándose que durante este año se incrementarán sus ingresos hasta un 807 %, desde los 1.411 millones de euros hasta los 11.496 millones de euros en 2013 [1]. Existe una amplia variedad de categorías de aplicaciones: desde herramientas de productividad, de monitorización del dispositivo, de fotografía, de educación, juegos, hasta herramientas de adquisición y canje de tiques de cualquier tipo.

Ya en 2002 apareció el primer estándar del Near Field Communication (NFC), una tecnología inalámbrica que permitía el intercambio de información entre dos dispositivos sin emparejamiento previo. En sus inicios esta tecnología se utilizó principalmente para leer tarjetas con los sistemas FeliCa y MIFARE.

Sin embargo, en los últimos años esta tecnología se ha empezado a incluir en los teléfonos móviles dotando a estos dispositivos de nuevas características.

Los desarrolladores aprovechan todas las funcionalidades que ofrecen los *smartphones* para crear nuevas aplicaciones que les permitan innovar en los aspectos más típicos de la vida cotidiana como por ejemplo la adquisición y empleo de un tique de transporte público. El sistema propuesto en este proyecto pretende crear una aplicación que permita agilizar este proceso de forma que sea más efectivo y cómodo para los clientes utilizando sus teléfonos móviles junto con la tecnología NFC.

1.2. Objetivos

Como se ha comentado en el apartado anterior, el objetivo de este Proyecto de Fin de Carrera es el diseño y desarrollo de un sistema que permita la adquisición y uso de tiques de transporte público a través de un teléfono móvil con sistema operativo Android y que disponga de tecnología Near Field Communication (NFC). Los objetivos a cumplir son:

- Estudio de los sistemas existentes y sus características (aplicaciones y/o proyectos piloto en evaluación)
- Concepción del sistema.
- Adquisición de los conocimientos necesarios sobre el sistema operativo Android que permitan la implementación de una aplicación para dispositivos móviles.
- Familiarización con la tecnología Near Field Communication.
- Búsqueda de librerías que permitan trabajar con lectores NFC.
- Estudio sobre la implementación de Servicios Web.
- Diseño de la aplicación móvil de manera que sea robusta y segura.
- Diseño del servidor que permite gestionar y almacenar todo la información relativa a los tiques.
- Implementación de un prototipo del sistema.

Actualmente, cuando un usuario desea utilizar cualquier medio de transporte público necesita adquirir un tique bien previamente (cercanías o metro) o bien en el mismo instante (autobús) que se accede al transporte. Este proceso de compra puede ser lento en función del número de pasajeros. El sistema que se propone pretende agilizar este proceso permitiendo la adquisición de los tiques de una forma sencilla y rápida de manera que el usuario únicamente deba validar el tique comprado frente a un lector que facilitara el acceso.

La principal ventaja del sistema que se propone en este documento frente al sistema tradicional de compra es la capacidad de adquirir los tiques en cualquier momento a través del dispositivo y así evitar colas en la estación. Otra ventaja que ofrece este sistema viene de la mano de la tecnología NFC y es el mecanismo

de uso del tique: simplemente bastará con que el usuario seleccione en su teléfono móvil que tique desea usar y pase el dispositivo por el lector que permite el acceso.

1.3. Metodología

Una de las tareas principales a la hora de crear una prueba de concepto es la selección de la metodología de desarrollo que se va a emplear. Por metodología se entiende una colección de procedimientos, técnicas, herramientas y documentos auxiliares que ayudan a los desarrolladores de software en sus esfuerzos por implementar nuevos sistemas de información.

En la actualidad existen dos grandes corrientes de metodologías de desarrollo de software. Por un lado, las denominadas metodologías tradicionales, centradas en el control del proceso, con un riguroso seguimiento de las actividades involucradas en ellas. Por otro lado, las metodologías ágiles, centradas en el factor humano, en la colaboración y participación del cliente en el proceso de desarrollo y a un incesante incremento de software con iteraciones muy cortas.

En este proyecto, al tratarse de un proyecto realizado por una sola persona, las metodologías tradicionales para el desarrollo de proyectos de software quedan descartadas puesto que están pensadas para grandes equipos de trabajo y no se adaptarían de forma adecuada. Además, estas metodologías se caracterizan por su estructura rígida y poco flexible a la hora de realizar cambios, el uso de documentos como método de intercambio de ideas y un control prácticamente total de todas las etapas de desarrollo.

Por esta razón, una metodología ágil se adecúa más a este proyecto teniendo en cuenta que solo se mantendrán las reuniones con el jefe del equipo, representado por el tutor del proyecto, y se omitirán las reuniones con los miembros del equipo por motivos obvios.

Se decide emplear una adaptación de la metodología Mobile-D creada como parte en un proyecto finlandés en 2005. Mobile-D es una mezcla de varias técnicas que combinadas suponen una contribución original para el escenario del desarrollo de aplicaciones para sistemas móviles. Esta metodología auna las prácticas de desarrollo de Extreme Programming (XP), la escalabilidad de la metodología Crystal y el ciclo de vida del Rational Unified Process (RUP).

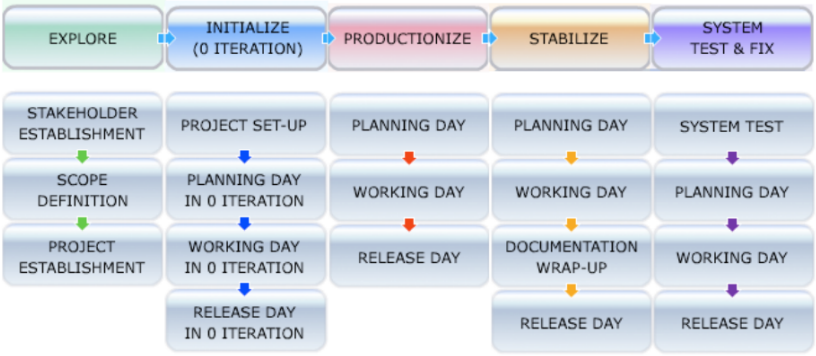

Figura 1.1: Esquema de la metodología Mobile-D

El ciclo del proyecto se divide en cinco fases: exploración, inicialización, *productización*, estabilización y prueba del sistema. En general, todas las fases (con la excepción de la primera fase exploratoria) contienen tres días de desarrollo distintos: planificación, trabajo y liberación.

La fase de exploración, siendo ligeramente diferente del resto del proceso de producción, se dedica al establecimiento de un plan de proyecto y los conceptos básicos. Por lo tanto, se puede separar del ciclo principal de desarrollo. En esta fase es importante la comunicación, en este caso, con el tutor del proyecto.

Durante la fase de inicialización se preparan e identifican todos los recursos necesarios. Se preparan los planes para las siguientes fases y se establece el entorno técnico.

En la fase de *productización* se repite la programación de tres días (planificación-trabajo-liberación). Se repite iterativamente hasta implementar todas las funcionalidades. Primero se planifica la iteración de trabajo en términos de requisitos y tareas a realizar. Las tareas se llevarán a cabo durante el día de trabajo desarrollando e integrando el nuevo código con el anterior. Durante el último día se lleva a cabo la integración del sistema seguida de las pruebas de aceptación.

En la fase de estabilización, se llevan a cabo las últimas acciones de integración para segurar que el sistema completo funciona correctamente. En esta fase, los desarrolladores realizarán tareas similares a las que debían desarrollar en la fase de "productización", aunque en este caso todo el esfuerzo se dirige a la integración del sistema. Adicionalmente se puede considerar en esta fase la producción de documentación.

La última fase (prueba y reparación del sistema) tiene como meta la disponibilidad de una versión estable y plenamente funcional del sistema. El producto terminado e integrado se prueba con los requisitos de cliente y se eliminan todos los defectos encontrados.

La adaptación realizada se muestra en la Figura 1.2. El esquema consta de las cinco etapas que componen la metodología Mobile-D: exploración, inicialización, *productización*, estabilización y prueba y reparación del sistema.

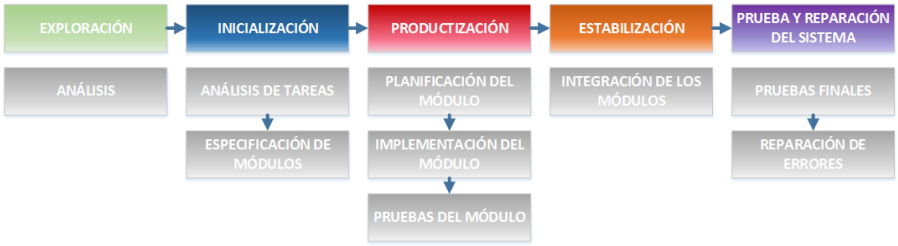

Figura 1.2: Esquema del ciclo de desarrollo utilizado

Durante la fase de exploración se realizará un análisis del estado del arte y de los principales problemas a decidir en el proyecto, especialmente aquellos que afecten a todo el sistema como son la arquitectura y los mecanismos de seguridad.

En el siguiente paso, durante la etapa de inicialización, se efectúa un análisis de las tareas que hay que realizar y se especifican los módulos que deben implementarse para que la aplicación cumpla la funcionalidad definida.

La tercera fase, *productización*, se repite iterativamente hasta finalizar todos los módulos del sistema. En cada una de esas iteraciones se lleva a cabo la planificación del módulo especificado, su implementación y finalmente se realizan una serie de pruebas sobre él para verificar su correcto funcionamiento.

Durante la etapa de estabilización se produce la integración de todos los módulos desarrollados. Hay que tener en cuenta que algún módulo puede depender de otro durante su implementación por lo que la integración de estos módulos se realizará con anterioridad.

Finalmente, durante la fase de prueba y reparación del sistema se realizan las pruebas finales del mismo como conjunto con las correspondientes correcciones en caso de que estas fueran necesarias.

1.4. Organización de este documento

El documento consta de varias partes que a su vez están formadas por varios capítulos y secciones en los que se explicará en detalle todos los aspectos más importantes del proyecto. En esta sección se va a explicar cómo se ha estructurado el documento.

En la Parte I se explica la motivación que dio lugar al proyecto y los objetivos que se desean alcanzar durante el desarrollo del mismo.

En la Parte II se desarrolla el análisis inicial del problema.

El Capítulo 2 presenta la situación actual del problema que el proyecto pretende resolver. Se describen las tecnologías usadas para desarrollar el proyecto, Near Field Communication (NFC) y el sistema operativo Android, así como otras aplicaciones o proyectos piloto relacionados con el sistema que se va a implementar.

En el Capítulo 3 se realiza una descripción de la funcionalidad que debe implementar la aplicación, que quedará recogida con detalle en una lista de requisitos de usuario, casos de uso y requisitos software.

La Parte III trata del diseño del prototipo, incluyendo diagramas de clases y modelos de la base de datos de todos los sistemas desarrollados. La información se divide en capítulos correspondientes con las distintas partes que conforman el sistema como un todo.

El Capítulo 4 presenta la arquitectura global del sistema y muestra un diagrama de despliegue que indica tanto el hardware como el software necesario en cada una de las partes del sistema para su correcto funcionamiento.

Los capítulos 5, 6 y 7 describen en detalle el diseño de la aplicación móvil, del servidor y de la aplicación del lector respectivamente, incluyendo el diseño de cada aplicación, clases que componen cada una de ellas, bases de datos necesarias en el caso de la aplicación móvil y el servidor, interfaces de usuario utilizadas en la aplicación móvil y una explicación de las decisiones de diseño tomadas durante el desarrollo del proyecto.

La Parte IV contiene en el Capítulo 8 una serie de mejoras que se considera que podrían aplicarse al sistema presentado con el fin de ampliar y/o mejorar su funcionalidad así como las conclusiones personales sobre el proyecto.

En el Capítulo 9 se muestra detallado el presupuesto total del proyecto.

Por último, se incluye un glosario con los términos más relevantes y la bibliografía y referencias citadas a lo largo de todo el documento.

Parte II

Análisis del problema

2

Estado del arte

2.1. Near Field Communication

2.1.1. Historia de Near Field Communication

Near Field Communication (NFC) es una tecnología inalámbrica, de corto alcance y alta frecuencia que permite el intercambio de información entre dos dispositivos de manera instantánea sin necesidad de emparejamiento previo. NFC tiene sus raíces en la tecnología de identificación por radiofrecuencia RFID (Radio Frecuency Identificacion - ISO 14443) [2].

La tecnología Near Field Communication fue desarrollada por Sony y Philips Electronics con el objetivo de diseñar un estándar que permitiera unificar los sistemas creados por ambas empresas: FeliCa [3] y MIFARE [4], respectivamente. En 2002 nace el primer estándar bajo el nombre de *mobile RFID* o *NFC*, realizado por el organismo ECMA (ECMA-340) [5]. Pero en 2004 llegaría el evento más importante en el proceso de estandarización de la tecnologóa NFC: Nokia, Philips y Sony se unen para crear el Near Field Communication Forum (NFC Forum) [6] con la finalidad de informar a los mercados y promover el uso avanzado de esta tecnología, desarrollando especificaciones y asegurando la interoperabilidad entre dispositivos y sevicios. A día de hoy, el NFC Forum ha elaborado distintas especificaciones entre las que cabría destacar las especificaciones NFCIP-1 y NFCIP-2, que definen el protocolo de comunicación entre dos dispositivos NFC (peer-to-peer). El NFC Forum también ha diseñado el logotipo N-Mark™como símbolo universal de la tecnología NFC permitiendo a los usuarios identificar fácilmente los productos que disponen de NFC y los lugares donde hay servicios NFC disponibles.

Figura 2.1: N-Mark™

En 2006 llegaría la especificación inicial de las etiquetas NFC. Una etiqueta NFC es un objeto pequeño, similar a una pegatina, que contiene información que un dispositivo con NFC puede leer. La información contenida en estas etiquetas suele ser de solo lectura, pero algunas etiquetas permiten al dispositivo sobreescribirla o modificarla. Ese mismo año también se elaboró la especificación para los *smart posters* que contienen información que puede ser leída por un dispositivo compatible con NFC. Un *smart poster* puede utilizarse en museos para proporcionar información de interés sobre una determinada obra de arte o sobre el autor, en carteles publicitarios para que los usuarios obtengan cupones de descuento electrónicos, sobre marquesinas de transporte público para que los usuarios puedan consultar los horarios...

El primer teléfono móvil que incorpora la tecnología NFC, el Nokia 6131, también surgió en la misma época. Con el paso de los años, aparecieron nuevas especificaciones y el uso de NFC se extendió a otros ámbitos como métodos de pago o compartición de contenidos entre dispositivos compatibles. En 2010 salió al mercado el primer teléfono Android con NFC, el Samsung Nexus S. El mercado NFC se está extendiendo rápidamente en Europa, Asia (especialmente en Japón) y en Estados Unidos. Se estima que en este último la tecnología NFC se convertirá con el paso del tiempo en una forma popular de pago e intercambio de información.

2.1.2. Características de Near Field Communication

Near Field Communication es una tecnología de comunicación inalámbrica de corto alcance (unos 20 cm) que permite el intercambio de información entre dos dispositivos. Esta tecnología funciona en la frecuencia de los 13.56 MHz, por lo que no se aplica ninguna restricción y no es necesaria una licencia para su uso. El estándar NFC es una extensión de las etiquetas RFID y se encuentra definido en la ISO/IEC 18092:2004 [7]. Un dispositivo NFC puede ser usado de dos modos:

- **Modo activo:** El dispositivo posee una fuente de energía que permite generar un campo electromagnético propio que posibilitará la transmisión de los datos.

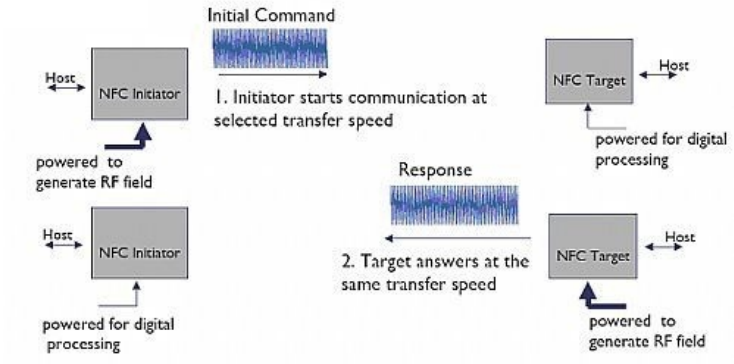

Figura 2.2: NFC: Modo Activo

- **Modo pasivo:** El dispositivo obtiene la energía del campo electromagnético generado por el dispositivo en modo activo.

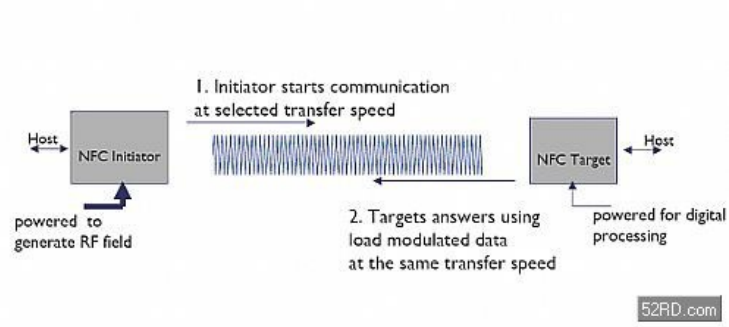

Figura 2.3: NFC: Modo Pasivo

Cuando dos dispositivos NFC se comunican pueden darse tres configuraciones posibles:

Dispositivo A	**Dispositivo B**	**Descripción**
Modo Activo	Modo Activo	Cuando un dispositivo envía datos genera un campo electromagnético. Si el dispositivo permanece a la espera de recibir datos no activará el campo electromagnético. De esta manera, el campo electromagnético será generado alternativamente por el dispositivo A y el dispositivo B.
Modo Activo	Modo Pasivo	El campo electromagnético es generado únicamente por el dispositivo A.
Modo Pasivo	Modo Activo	El campo electromagnético es generado únicamente por el dispositivo B.

Tabla 2.1: Configuraciones de comunicación NFC

Además de los modos activo y pasivo, existen dos roles que pueden desempeñar los dispositivos durante la comunicación NFC: iniciador y receptor. El iniciador es el dispositivo que comienza la comunicación y controla el intercambio de datos. Por otro lado, el receptor es el dispositivo que responde a las peticiones del iniciador. En la siguiente tabla (Tabla 2.2) se muestran las posibles combinaciones que pueden darse entre los modos activo/pasivo, y los roles de iniciador/receptor.

El modo en el que se encuentra configurado el dispositivo es importante ya que define como se lleva a cabo la transmisión de los datos. Los datos se envían utilizando modulación por desplazamiento de amplitud (del inglés Amplitude Shift Keying, ASK) [8] y soporta velocidades de transmisión de datos de 106 kbits/s, 212 kbits/s y 424 kbits/s. En modo activo, si la tasa de datos es de 106

	Iniciador	**Receptor**
Modo Activo	Posible	Posible
Modo Pasivo	No posible	Posible

Tabla 2.2: Posibles combinaciones Activo/Pasivo con Iniciador/Receptor

kbits/s se emplea un esquema de codificación de Miller modificado [9] con un 100 % de modulación. En cualquier otro, ya sea modo activo o pasivo, caso se utiliza una codificación Manchester [10] con una modulación del 10 %.

El estándar NFC se construyó con el objetivo de ser utilizado en teléfonos móviles pero su uso se está extendiendo a otros dispositivos móviles como tabletas (Google Nexus 7, Google Nexus 10) y videoconsolas (Wii U). El estándar permite el usar la tecnología NFC de tres formas diferentes:

- **Card Emulation:** El dispositivo funciona de la misma manera que una *smartcard*.
- **Reader/Writer:** El dispositivo puede leer o escribir etiquetas RFID.
- **Peer to Peer:** Ambos dispositivos pueden intercambiar información.

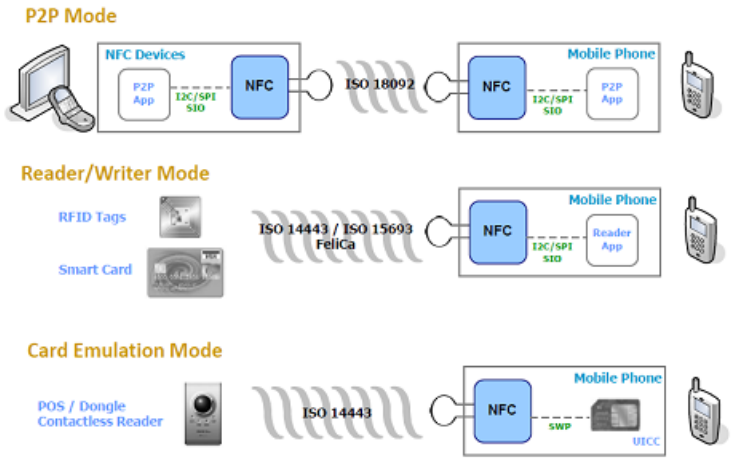

Figura 2.4: Modos de operación de NFC

Una transacción NFC siempre sigue una misma secuencia de operación que consta de los siguientes pasos:

1. **Descubrimiento:** El iniciador y el receptor están dentro del campo de acción.
2. **Autenticación:** Procedimiento seguro de autenticación y uso de mecanismos anticolisión.
3. **Negociación:** En esta fase se establecen los parámetros de la comunicación tales como la velocidad de transmisión, identificador del dispositivo,

tipo de aplicación, tamaño de la transferencia y acción que se desea llevar a cabo.

4. **Transferencia:** Intercambio de datos entre los dispositivos.

5. **Reconocimiento:** El receptor confirma la recepción de los datos.

2.1.3. Mensaje NFC Data Exchange Format (NDEF)

El NFC Forum definió el formato de encapsulación de los mensajes que permitirían el intercambio de información entre dos dispositivos NFC y lo denominó NFC Data Exchange Format (NDEF). NDEF es un mensaje ligero que permite encapsular uno o varios *payloads* de distinto tipo y/o tamaño. Algunos de los tipos de *payloads* son URIs (Uniform Resource Identifier), MIME (Multipurpose Internet Mail Extension) o otros tipos específicos de NFC.

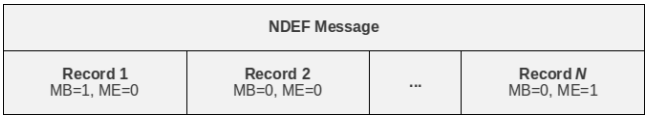

Figura 2.5: Mensaje NDEF

Un mensaje NDEF está formado por uno o más NDEF *Records* o registros NDEF. El primer registro tiene activado el flag MB (Message Begin) que indica el inicio de un mensaje NDEF. El último registro tendrá activado el flag ME (Message End) que indica el final de un mensaje NDEF. En caso de que un mensaje NDEF estuviera formado por un único registro, este registro tendría activados ambos flags. El número de registros que pueden componer un mensaje NDEF es ilimitado.

NDEF *Record*

Como ya se comento en el apartado anterior, un mensaje NDEF está formado por uno o más registros NDEF. Cada registro NFC contiene un *payload* con un tamaño de hasta $2^{32} - 1$ octetos. Un registro NDEF incluye tres parámetros que describen el *payload*:

- **PAYLOAD_LENGTH:** Indica el número de octetos del *payload*.

- **TYPE:** Indica el tipo *payload*. Este campo permite que el *payload* sea tratado por la aplicación de usuario adecuada.

- **ID:** Este parámetro tiene la forma de una URI absoluta o relativa que permite enlazar con otros *payloads* para realizar referencias cruzadas.

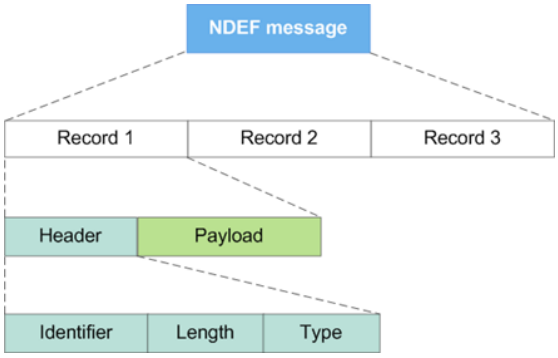

Figura 2.6: Formato de mensaje NDEF

Además de estos parámetros, un registro NDEF incluye los siguientes campos:

- **MB (Message Begin):** Campo de un bit que si esta activo indica el inicio de un mensaje NDEF.
- **ME (Message End):** Campo de un bit que si esta activo indica el final de un mensaje NDEF.
- **CF (Chunk Flag) :** Campo de un bit que si esta activo indica bien si es el primer registro fragmentado o bien si es un registro intermedio dentro del *payload* fragmentado.
- **SR (Short Record):** Campo de un bit que si esta activo indica que el campo PAYLOAD_LENGTH ocupa únicamente un octeto en lugar de cuatro. Este campo permite una encapsulación compacta para *payloads* pequeños (tamaño entre 0 y 255 octetos). Un registro puede contener tanto *payloads* cortos como *payloads* normales.

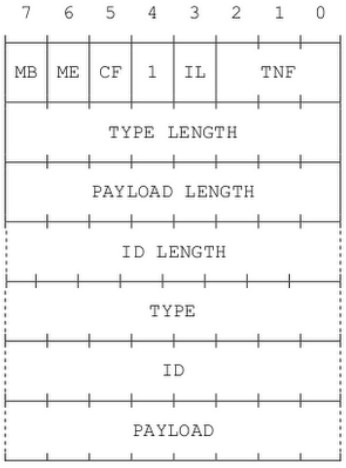

Figura 2.7: NDEF Short Record

- **IL (ID_LENGTH field is present):** Campo de un bit que si esta activo indica que el parámetro ID_LENGTH está presente en la cabecera como un único octeto. Si el flag no está activado, el campo ID_LENGTH se omite en la cabecera y el campo ID se omite también en el registro.
- **TNF (Type Name Format):** Campo de 3 bits que indica la estructura del contenido del parámetro TYPE. Puede tomar los siguientes valores:

Type Name Format	Value
Empty	0x00
NFC Forum well-known type [NFC RTD]	0x01
Media-type as defined in RFC 2046 [RFC 2046]	0x02
Absolute URI as defined in RFC 3986 [RFC 3986]	0x03
NFC Forum external type [NFC RTD]	0x04
Unknown	0x05
Unchanged	0x06
Reserved	0x07

Figura 2.8: Valores del campo TNF

- **TYPE_LENGTH:** Campo que contiene un entero sin signo de 8 bits que especifica la longitud en octetos del campo TYPE.
- **ID_LENGTH:** Campo que contiene un entero sin signo de 8 bits que especifica la longitud en octetos del campo ID. Este campo está presente solamente si el flag IL está activo.

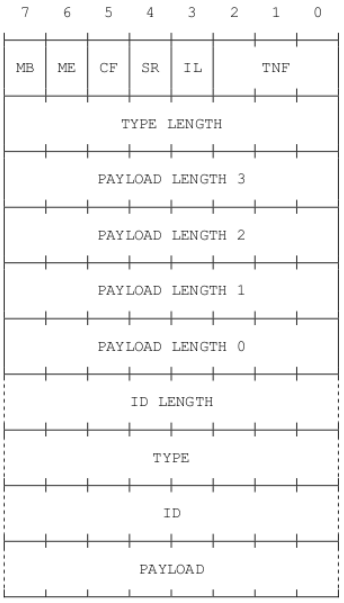

Figura 2.9: NDEF Record

2.1.4. Simple NDEF Exchange Protocol (SNEP)

El *Simple NDEF Exchange Protocol* (SNEP) es un protocolo a nivel de aplicación que permite el envío y recepción de unidades de datos de aplicación en forma de mensajes NDEF entre dos dispositivos NFC operando en modo peer-to-peer. Este protocolo hace uso del *Logical Link Control Protocol* (LLCP) orientado a conexión en modo transporte para proporcionar un intercambio de datos fiable. LLCP define un protocolo OSI de capa de enlace que permite la comunicación bidireccional entre dos dispositivos NFC en modo peer-to-peer. Es un protocolo compacto basado en el estándar IEEE 802.2 que ofrece una base sólida para aplicaciones peer-to-peer y que mejora la funcionalidad básica ofrecida por la norma ISO 18092.

El protocolo SNEP es un protocolo que sigue el esquema de petición/respuesta.

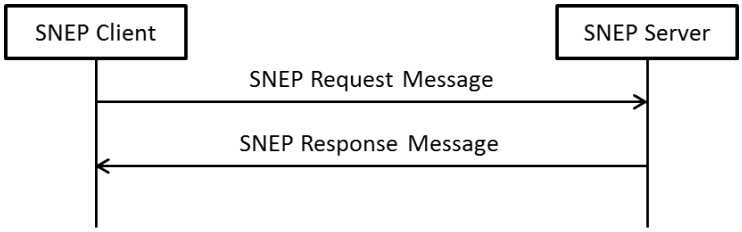

Figura 2.10: Modelo de comunicación del protocolo SNEP

Un cliente SNEP envía una petición a un servidor SNEP que contiene la versión del protocolo, el método de la petición (put o get), la longitud del campo información en octetos y el campo información. El servidor SNEP lleva a cabo la acción indicada en la petición usando la información recibida y responde con un mensaje que contiene la versión del protocolo, un código de estado que indica el éxito o fallo de la petición, la longitud del campo información en octetos y el campo información.

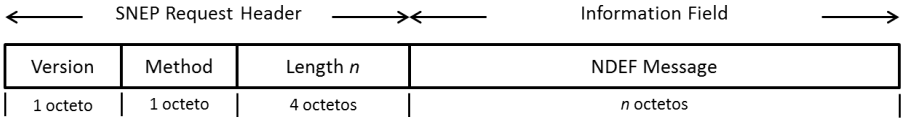

Figura 2.11: Petición SNEP

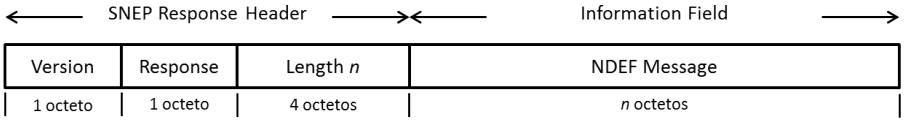

Figura 2.12: Respuesta SNEP

El propósito principal del protocolo SNEP es el intercambio de mensajes NDEF. Los mensajes NDEF serán transmitidos en el campo información de los

mensajes de petición y respuesta. La longitud máxima de los mensajes NDEF transmitidos es de $2^{32} - 1$ octetos. Si la longitud total del mensaje petición o respuesta SNEP excede este tamaño es posible fragmentar dicho mensaje para su transmisión. Los fragmentos se construyen de manera que los octetos enviados sean consecutivos hasta completar el mensaje original. Para que el receptor pueda determinar el número de octetos que conforman el mensaje completo el primer fragmento debe contener la cabecera completa de mensaje SNEP.

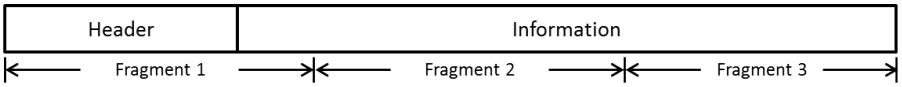

Figura 2.13: Fragmentación de mensaje SNEP

Tras recibir el primer fragmento, el receptor de un mensaje SNEP fragmentado debe indicar al emisor que puede recibir el resto de fragmentos. Los fragmentos restantes serán enviados por el receptor una vez que recibe la indicación de continuar por parte del receptor. La recepción de los fragmentos restantes no serán confirmados por parte del receptor.

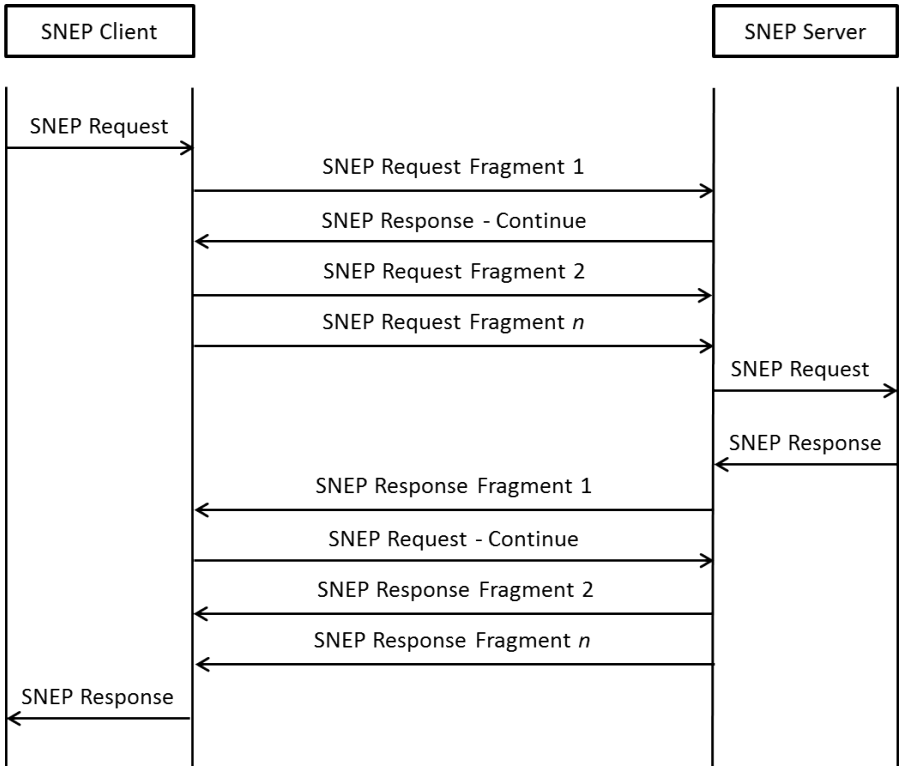

Figura 2.14: Intercambio de mensaje SNEP fragmentado

2.1.5. Seguridad en Near Field Communication

Al ser una tecnología de comunicación inalámbrica, el principal punto débil de NFC en el ámbito de la seguridad se encuentra en el canal por el que se transmite la información: el aire. Hay que tener en cuenta que NFC todavía es una tecnología en periodo de pruebas por lo que todavía existen posibles vectores de ataque para los que todavía no se ha aclarado la manera en que serán asegurados. En los próximos años, en los que se considera que el uso de NFC se tornará masivo será cuando se empiecen a conocer las verdaderas vulnerabilidades tanto de la tecnología como de los dispositivos que la integran.

En la actualidad, los análisis de la seguridad NFC únicamente pueden basarse en los estudios realizados por investigadores como el que realizan Ernst Haselsteiner y Klemens Breitfuß [11] o documentos como el que presenta el Instituto Nacional de Tecnologías de la Comunicación (Inteco) [12].

Interceptación o *eavesdropping* Cuando dos dispositivos se comunican haciendo uso de la tecnología NFC se utilizan ondas electromagnéticas que pueden ser capturadas por un atacante que cuente con el equipamiento necesario. La comunicación NFC está pensada para realizarse entre dos dispositivos muy próximos entre sí (unos 10cm o menos) por lo que la principal cuestión es cómo de cerca debe encontrarse un atacante para capturar la señal de forma clara. Esta distancia viene determinada por una serie de parámetros como: características del campo electromagnético del dispositivo que envía, características de la antena del atacante, calidad del receptor del atacante, calidad del decodificador de señal del atacante, obstáculos del entorno... También es importante tener en cuanta el modo (activo/pasivo) en el que opera el dispositivo que envía la información ya que es mucho más complicado capturar la señal de un dispositivo en modo pasivo. Si el dispositivo que envía está en modo activo, la interceptación de la señal se puede hacer a una distancia de unos 10m. Por otro lado, si el dispositivo está en modo pasivo la distancia de interceptación se ve considerablemente reducida a 1m. Como ocurre en otros esquemas de comunicación, la mejor manera de evitar la interceptación y proteger la información transmitida es establecer un canal seguro.

Corrupción de datos Además de interceptar la señal, un atacante puede intentar modificar los datos transmitidos. En el supuesto más simple, el atacante corromperá la información que se está transmitiendo vía NFC de manera que el receptor no sea capaz de entender los datos enviados por el otro dispositivo provocando así un ataque de denegación de servicio (DoS). Para ello, el atacante debe transmitir datos en frecuencias válidas y en el momento adecuado. Esta información puede ser calculada por el atacante si conoce bien el esquema de la modulación y la codificación. Sin embargo, los dispositivos NFC pueden verificar el campo electromagnético mientras transmiten información por lo que no sería demasiado complicado detectar este tipo de ataques.

Modificación de datos Mediante esta técnica el atacante es capaz de modificar los datos enviados por un dispositivo de manera que el receptor reciba información válida pero manipulada. En este caso, es necesario que el atacante

conozca la codificación usada por el emisor para poder modificar todos o únicamente algunos bits del mensaje. La viabilidad de este método depende en gran medida de de la exactitud en la modulación de la señal emitida por el atacante.

Inserción de datos En este caso, el atacante trata de insertar mensajes completos en la comunicación entre dos dispositivos suplantando al receptor. Esto solo sería posible en casos donde el dispositivo receptor necesite mucho tiempo para responder, de manera que el atacante utilice ese tiempo para insertar su mensaje. Si el mensaje original y el mensaje enviado por el atacante se solapan, los datos se corromperán. Al igual que ocurre con la interceptación, la mejor manera de protegerse frente a este tipo de ataques es establecer un canal de comunicación seguro.

Ataque *Man in the Middle* En un ataque *Man in the Middle*, el atacante intercepta la comunicación entre dos sistemas de manera que pasa a través de él. Básicamente, los dispositivos originales creen estar comunicándose entre sí mientras que lo hacen a través del atacante que puede manipular la información que se transmite. En el contexto de la comunicación NFC pueden darse dos situaciones:

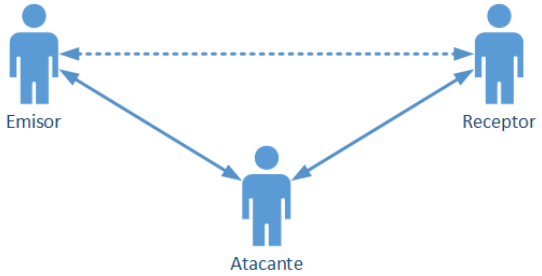

Figura 2.15: Ataque Man in the Middle

- **Comunicación modo activo - modo pasivo:** Asumiendo que el emisor usa modo activo y el receptor usa modo pasivo, el atacante deberá interceptar la información enviada por el emisor así como asegurarse de que al receptor no le llegan estos datos corrompiendo la comunicación. Esta situación puede ser detectada por el emisor, ya que los dispositivos NFC cuentan con detector de colisiones, que finalizará inmediatamente la comunicación. Si el emisor no detectara esta situación, el atacante debe enviar un mensaje al receptor. En este punto surge el segundo problema, para poder enviar datos el atacante debe generar un campo electromagnético pero el emisor mantiene el suyo activo. La única solución que tiene el atacante es alinear perfectamente su campo electromagnético con el del emisor. Esto es prácticamente imposible por lo que es muy complicado que pueda darse un ataque de *Man in the Middle* en esta situación.
- **Comunicación modo activo - modo activo:** Igual que ocurría en el caso anterior, en primer lugar el atacante deberá interceptar la información enviada por el emisor y asegurarse de que no le llegan datos al receptor sin

ser detectado en el proceso. Si no es detectado, el atacante puede enviar su mensaje al receptor sin inconvenientes, ya que el emisor ha apagado su campo electromagnético. El problema surge porque el emisor también estará a la espera de recibir un mensaje y le llegarán los datos enviados por el atacante. El emisor comprobará que la respuesta recibida no es válida y cerrará la comunicación. Por lo que un ataque de *Man in the Middle* con está configuración también es bastante improbable.

Redirección a sitios maliciosos a través de etiquetas NFC Con anterioridad se han detectado ataques de *spoofing* de sitios web que contenían *exploits* del navegador del móvil a través de códigos QR y esta técnica podría repetirse con etiquetas NFC. Al leer una etiqueta no se sabe de antemano la URI a la que dirige y es posible que redirija a sitios web con ataques de tipo *cross-site scripting*, *exploits* o cualquier otro uso fraudulento conocido basado en web (*phising*, troyanos...). Debido a esto se incluyó la Definición de Tipo de Registro (del inglés, *Record Type Definition* (RTD)) firma que permite a los fabricantes firmar el contenido de un campo URI. De esta manera, es posible conocer si el sitio web al que dirige la etiqueta es de confianza.

2.2. Android

Android, Inc. fue fundado en 2003 por Andy Rubin (cofundador de Danger Inc.), Rich Miner (cofundador de Wildfire Communication Inc.), Nick Sears (vicepresidente de T-Mobile) y Chris White (a cargo del diseño y y desarrollo de interfaces en WebTV) con el objetivo de desarrollar dispositivos móviles inteligentes que fueran capaces de adecuarse a la localización y las preferencias del usuario. A pesar de los éxitos pasados de sus creadores, en esa época Android era relativamente desconocido y funcionaba prácticamente en secreto bajo la consigna de crear un software para teléfonos móviles.

En julio de 2005, Google adquiere Android Inc. manteniendo a los empleados clave de la compañía. Con el apoyo de los recursos de Google, el ritmo de desarrollo de Android se vió incrementado notablemente y el equipo dirigido por Rubin desarrolló una plataforma para teléfonos móviles que funcionaba sobre el kernel de Linux. Google promocionó la plataforma entre fabricantes de teléfonos y operadores móviles con la promesa de proveer un sistema flexible y actualizable.

En noviembre de 2007, la Open Handset Alliance [13], un consorcio de empresas de tecnología como Google, fabricantes de dispositivos como HTC y Samsung, operadores como Sprint Nextel y T-Mobile y fabricantes de chips como Qualcomm y Texas Instruments, se dio a conocer con la finalidad de desarrollar estándares abiertos para dispositivos móviles. Una semana después, Google presentó una primera versión de Android, una plataforma para teléfonos móviles basada en la versión 2.6 del kernel de Linux y lanzó la primera versión del Android Software Development Kit (SDK). No obstante, hasta septiembre de 2008 no se puso a la venta el primer teléfono móvil con Android, el HTC Dream.

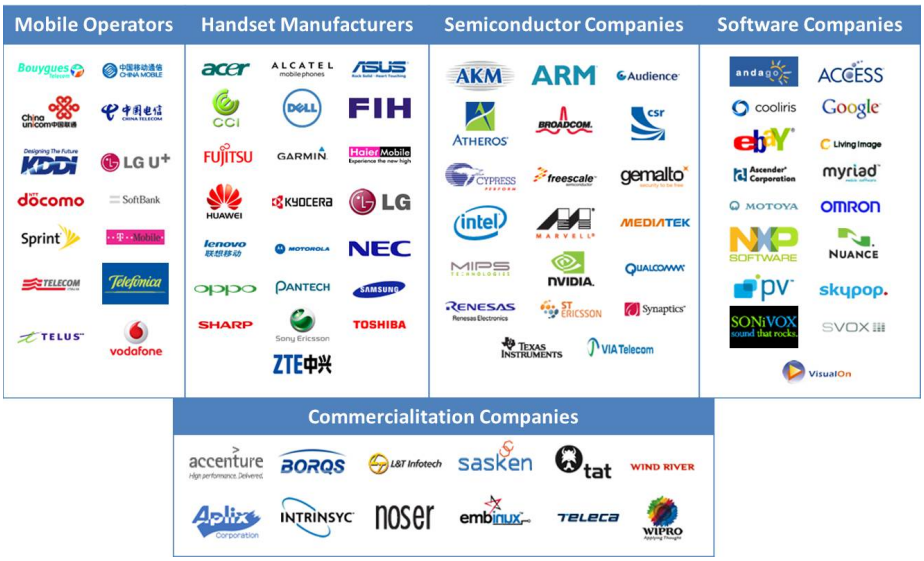

Figura 2.16: Open Handset Alliance

En agosto de 2008, Google anuncia el Android Market donde los desarrolladores podrían subir sus aplicaciones para que todos los usuarios pudieran descargarlas. En un principio no estaba soportado el sistema de pago por las aplicaciones que sería posteriormente introducido en 2009.

2.2.1. Arquitectura

Para comprender el funcionamiento del sistema operativo Android es necesario entender como está estructurado y que componentes lo forman. La arquitectura de Android está formada por varias capas que facilitan el desarrollo de aplicaciones para la plataforma tal y como se muestra en la Figura 2.17. Cada una de las capas utiliza elementos de la capa inferior para realizar sus funciones, es por ello que a este tipo de arquitectura se le conoce también como pila. Este modelo hace posible que los desarrolladores puedan acceder a las capas más bajas del sistema operativo, que permiten usar los componentes hardware de los teléfonos, a través de librerías.

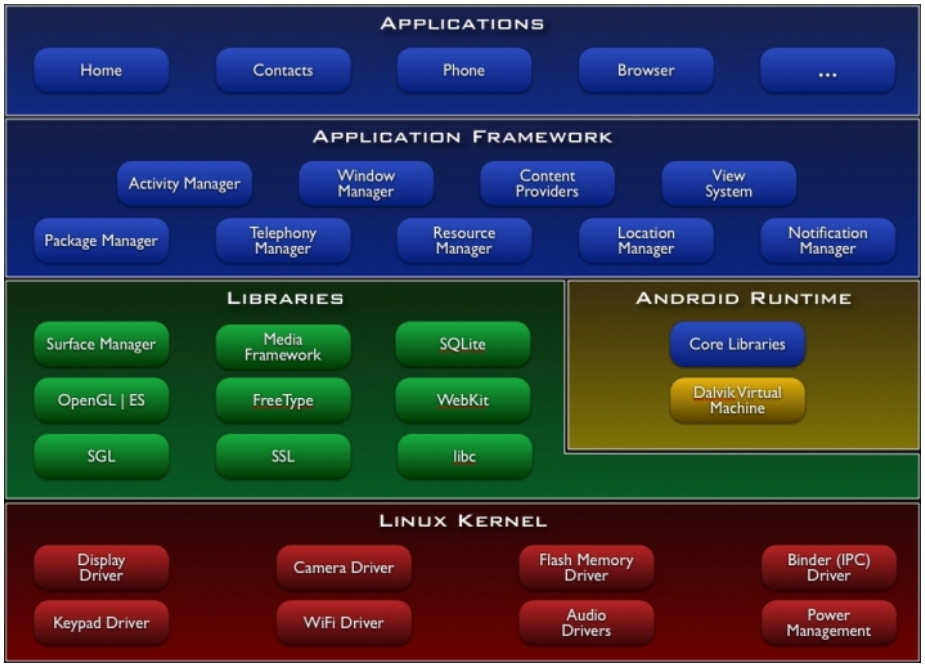

Figura 2.17: Arquitectura de Android

Kernel de Linux El núcleo del sistema operativo Android está basado en la versión 2.6 del kernel de Linux que administra la seguridad, la memoria, los procesos, la pila de red y los drivers, proporcionando las necesidades básicas de software para arrancar y gestionar tanto el hardware como las aplicaciones. El núcleo actúa como una capa de abstracción entre el hardware y el resto de las capas de la arquitectura. El kernel de Android está disponible de manera libre en http://android.git.kernel.org/. Sin embargo, el kernel solo suele ser modificado por fabricantes de hardware y dispositivos que quieren asegurarse de que el sistema operativo funcione correctamente en sus dispositivos.

Librerías Esta capa se sitúa sobre el kernel y esta formada por librerías nativas de Android escritas en C o C++ que han sido compiladas para la arquitectura hardware específica del teléfono. Estas librerías son desarrolladas normalmente por los fabricantes de dispositivos. Algunas de las librerías más importantes son OpenGL (motor gráfico), Media Framework (códecs de formatos de audio, imagen y vídeo), Webkit (navegador), SSL (cifrado de comunicaciones), FreeType (fuentes de texto) y SQLite (base de datos).

Entorno de ejecución de Android El entorno de ejecución de Android no es considerado una capa por sí mismo ya que está formado por librerías. El entorno de ejecución está formado por dos componentes: la máquina virtual *Dalvik* y las librerías del núcleo de Java. La máquina virtual *Dalvik* es un tipo de máquina virtual Java optimizado para sistemas que están limitados en términos de procesamiento y memoria. Las aplicaciones están programadas normalmente en Java y son compiladas en *bytecode*. El bytecode resultante se transforma

entonces en un formato compatible con la máquina virtual Dalvik (.dex) para su posterior instalación en el dispositivo. La máquina virtual Dalvik permite la creación de múltiples instancias simultáneamente proporcionando seguridad, aislamiento, gestión de memoria y soporte para hilos. Al ser una variante de la máquina virtual de Java permite la instalación de aplicaciones en los dispositivos independientemente del hardware siempre que dispongan de la versión mínima de Android requerida. Las librerías del núcleo de Java son diferentes de las librerías Java SE y Java ME pero proporcionan prácticamente la misma funcionalidad que la primera.

Framework de aplicaciones La siguiente capa está formada por todas las clases y servicios con los que interaccionan las aplicaciones. Estos componentes gestionan las funciones básicas del teléfono y acceden a recursos de las capas anteriores a través de la máquina virtual Dalvik. Algunos de los bloques más importantes de esta capa son: *Activity Manager*, que se encarga de gestionar las actividades de las aplicaciones así como sus ciclos de vida; *Content Provider*, que encapsula los datos que se comparten entre aplicaciones y permite mantener un control de acceso a dichos datos; *Views*, elementos que permiten construir las interfaces que se muestran al usuario (botones, listas, cuadros de texto...); *Telephony Manager*, que permite realizar/recibir llamadas y enviar/-recibir mensajes de texto; *Location Manager*, que hace posible la obtención de la localización del dispositivo mediante GPS o redes disponibles...

Aplicaciones Esta es la última capa de la arquitectura de Android. En esta capa se incluyen todas las aplicaciones del dispositivo, tanto las que tienen interfaz gráfica como las que no, las nativas (programadas en C/C++) y las administradas (programadas en Java), las preinstaladas y aquellas que ha instalado el usuario.

2.2.2. Máquina Virtual *Dalvik*

Como se mencionó en el apartado anterior, el sistema operativo Android utiliza una máquina virtual llamada *Dalvik* que se encuentra en la capa del entorno de ejecución y que ha sido especialmente diseñada para optimizar el uso de la memoria y los recursos de hardware en dispositivos móviles. *Dalvik* también está optimizada para permitir la ejecución de múltiples instancias de la máquina virtual simultáneamente con un impacto muy bajo en el rendimiento de la memoria del dispositivo. Este aspecto de usar varias máquinas virtuales se pensó para proteger a las aplicaciones, de forma que el cierre o fallo inesperado de alguna de ellas no afecte de ninguna forma a las demás.

La máquina virtual *Dalvik* fue diseñada por Dan Bornstein con contribuciones de otros ingenieros de Google. Recibe su nombre en honor a Dalvík, un pueblo de Eyjafjörour, Islandia, donde vivieron antepasados de Bornstein.

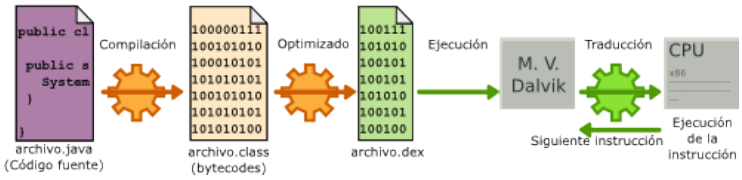

Figura 2.18: Proceso de compilación en Android

Dalvik es una máquina virtual que es gestionada y ejecutada como cualquier otra aplicación, pero su función es la de proporcionar un entorno de programación independiente de la plataforma que permita obviar el hardware instalado por debajo y posibilite la ejecución de los programas realizados para ella en cualquier tipo de plataforma sin tener que realizar modificación alguna en el código de la aplicación. Utiliza el modelo de compilación Just-In-Time (JIT), también conocido como traducción dinámica, que es un híbrido entre los lenguajes interpretados y los compilados y que consiste en traducir el *bytecode* a código máquina nativo en tiempo de ejecución, lo que mejora el rendimiento considerablemente.

El intérprete toma los archivos generados por las clases Java (.class) y los combina en uno o más archivos ejecutables *Dalvik* (.dex), los cuales a su vez son comprimidos en un sólo fichero .apk (Android Package) en el dispositivo. De esta forma, reutiliza la información duplicada por múltiples archivos .class, reduciendo así la necesidad de espacio (sin comprimir) a la mitad de lo que ocuparía un archivo .jar.

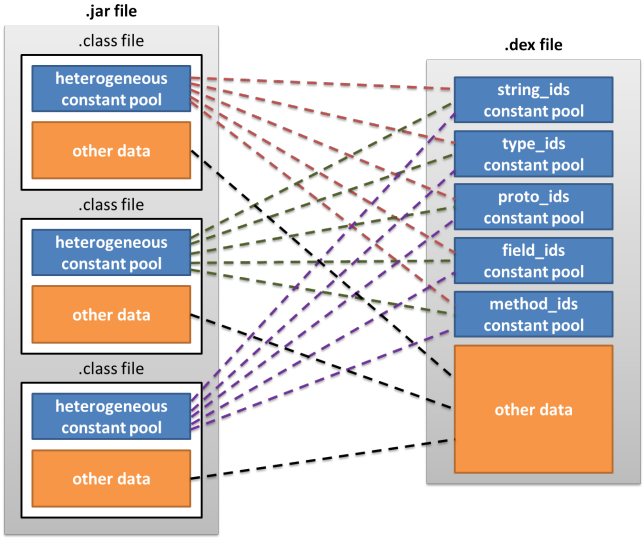

Figura 2.19: .jar VS .dex

A diferencia de la máquina virtual Java (JVM), que es una máquina virtual de pila, *Dalvik* es una máquina virtual de registro. Este tipo de máquinas virtuales tienen como modelo la máquina de *Turing* con uno o más registros

que sustituyen a la cinta y el cabezal utilizado en dicha máquina teórica. Las máquinas virtuales de pila utilizan una o más pilas como forma de utilizar la memoria de la máquina y su ventaja respecto a las máquinas virtuales de registro reside en que generalmente tienen una mayor densidad de código lo que facilita la lectura. Sin embargo, las máquinas virtuales de registro, al basarse en estos últimos, suelen ser más rápidas que las de pila dado que estas últimas se basan en memoria, aunque existen máquinas de pila que incluyen cachés en registros para acelerar la ejecución en la medida de lo posible.

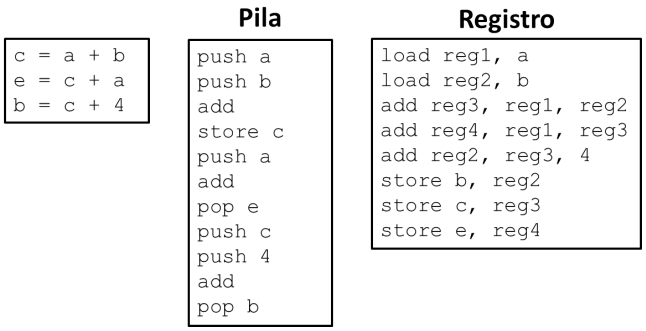

Figura 2.20: Pila VS Registro

2.2.3. Características de Android

Debido a que Android es un sistema operativo de código abierto (bajo licencia Apache) y está disponible libremente para que los fabricantes de dispositivos puedan adaptarlo a sus necesidades, Android no tiene una configuración predefinida ni de hardware ni de software. Sin embargo, Android por si mismo soporta las siguientes características:

- **Diseño:** La plataforma es adaptable a pantallas de mayor resolución, VGA, biblioteca de gráficos 2D, biblioteca de gráficos 3D basada en las especificaciones de la OpenGL ES 2.0 y diseño de teléfonos tradicionales.
- **Almacenamiento:** Android usa SQLite, una base de datos relacional ligera, para almacenar datos.
- **Conectividad:** Soporta GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth (A2DP and AVRCP), Wi-Fi, LTE, HSDPA, HSPA+ y WiMAX.
- **Mensajería:** Soporta SMS y MMS.
- **Navegador web:** El navegador web incluido está basado en el motor de renderizado de código abierto WebKit, junto con el motor JavaScript V8 de Google Chrome.
- **Soporte multimedia:** Soporta WebM, H.263, H.264 (en 3GP o MP4), MPEG-4 SP, AMR, AMR-WB (en un contenedor 3GP), AAC, HE-AAC (en contenedores MP4 o 3GP), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF y BMP.

- **Soporte para hardware:** Android soporta cámaras de fotos, de vídeo, pantallas táctiles, GPS, acelerómetros, giroscopios, magnetómetros, sensores de proximidad y de presión, sensores de luz, gamepad, termómetro, aceleración por GPU 2D y 3D.

- **Multitáctil:** Android tiene soporte nativo para pantallas capacitivas con soporte multi-táctil.

- **Multitarea:** Android soporta multitarea real de aplicaciones, es decir, las aplicaciones que no estén ejecutándose en primer plano reciben ciclos de reloj, a diferencia de otros sistemas de la competencia en la que la multitarea es congelada.

- ***Tethering*:** Permite al teléfono ser usado como un punto de acceso alámbrico o inalámbrico.

2.2.4. Versiones de Android

Google ha lanzado numerosas actualizaciones del sistema operativo Android desde su lanzamiento. La tabla 2.3 muestra la relación de todas las versiones de Android lanzadas hasta la fecha. Cada versión se identifica por un número y por un nombre de postre en inglés. En cada versión el postre elegido empieza por una letra distinta siguiendo un orden alfabético.

Versión de Android	Fecha de lanzamiento	Nombre	API
1.0		Apple Pie	
1.1	9 de febrero de 2009	Banana Bread	
1.5	30 de abril de 2009	Cupcake	3
1.6	15 de septiembre de 2009	Donut	4
2.0/2.1	26 de octubre de 2009	Eclair	7
2.2	20 de mayo de 2010	Froyo	8
2.3/2.3.2	6 de diciembre de 2010	Gingerbread	9
2.3.3/2.3.7	9 de febrero de 2011	Gingerbread	10
3.0/3.1	10 de mayo de 2011	Honeycomb	12
3.2	15 de julio de 2011	Honeycomb	13
4.0/4.0.4	16 de diciembre de 2011	Ice Cream Sandwich	15
4.1	9 de julio de 2012	Jelly Bean	16
4.2	13 de noviembre de 2012	Jelly Bean	17

Tabla 2.3: Versiones de Android

En septiembre de 2008 salió a la venta el primer dispositivo Android con la versión 1.0, la HTC Dream (Figura 2.21), que incluía una serie de servicios de Google como Gmail (correo), Calendar (calendario), Maps (mapas), Search (búsqueda), GTalk (mensajería instantánea) y otras características como YouTube, Android Market, WiFi, Bluetooth, notificaciones...

Figura 2.21: HTC Dream

En la versión 1.5 se incluyeron nuevas características como la capacidad para grabar vídeos en formato MPEG-4 y 3GP, el soporte para *Widgets* o la posibilidad de subir videos a Youtube y fotos a Picassa.

Un año más tarde aparecía Android 2.0 que incorporaba una experiencia mejorada en el Android Market, una mejora en la búsqueda por voz, actualización de soporte para CDMA/EVDO, 802.1x, VPN y *text-to-speech*. Las siguientes revisiones de esta versión optimizaban el rendimiento del sistema operativo y la gestión de memoria y mejoraban otros aspectos tales como ofrecer soporte para Microsoft Exchange, funcionalidad de WiFi hotspot, soporte para Adobe Flash 10.1, soporte nativo para más sensores como giroscopios y barómetros, soporte nativo para múltiples cámaras, soporte nativo para telefonía VoIP SIP...

En febrero de 2011, Google lanzó la versión 3.0 de Android que estaba dirigida únicamente a las tabletas e incluía una serie de mejoras. Las aplicaciones desarrolladas para versiones anteriores de Android eran compatibles con los dispositivos que tuvieran instalada está versión de Android pero no ocurría lo mismo en el caso contrario, es decir, las aplicaciones desarrolladas usando la API de la versión 3.0 de Android no funcionaban en dispositivos con versiones previas de Android.

Para solucionar esto, en octubre de 2011, Google lanzó la versión 4.0 de Android que traía todas las mejoras incorporadas en la versión 3.0 a los *smartphones* y además incluía nuevas características como desbloqueo mediante reconocimiento facial, monitorización y control del uso de datos, Near Field Communication (NFC)...

La Figura 2.22 presenta un gráfico que recoge la cuota de mercado de cada una de las versiones de Android a día de hoy:

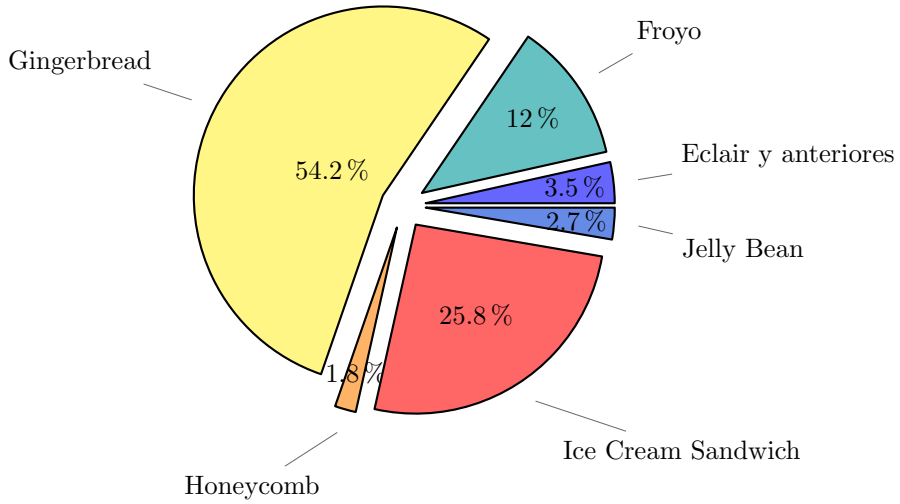

Figura 2.22: Distribución de versiones Android

2.2.5. Estructura de una aplicación Android

Las aplicaciones Android se escriben usando el lenguaje de programación Java. Las herramientas del Android SDK compilan el código (con los datos y los recursos necesarios) en un paquete Android, un archivo con la extensión .apk. Todo lo incluido en un archivo .apk conforma una aplicación Android y es lo que los dispositivos usaran para instalar la aplicación.

Los componentes de las aplicaciones son los bloques esenciales que permiten construir una aplicación Android. Cada componente refleja una forma diferente mediante la cual la aplicación interactua con el sistema. Hay cuatro tipos diferentes de componentes, cada uno de ellos tiene un propósito distinto y un ciclo de vida diferente que define como se crea el componente y cuando se destruye.

Activities Una actividad representa una pantalla con una interfaz de usuario. Por ejemplo, una aplicación de correo tendría una *activity* que mostraría una lista con los correos nuevos, otra *activity* que permitiría enviar correos y otra que permitiría leer los correos. A pesar de que las *activities* trabajan juntas para formar una experiencia cohesiva en la aplicación de correo desde el punto de vista del usuario, cada una es independiente de las otras.

Una *activity* puede encontrarse en distintos estados:

- **Activa o en ejecución:** Está la primera en la pila de ejecución, el usuario ve la actividad y puede interactuar con ella.
- **Pausada:** Cuando ha pasado a un segundo plano pero todavía es parcialmente visible. Típicamente ocurre cuando se abre un diálogo encima de la pantalla. En este caso, la actividad tapada puede ser cerrada por el sistema si necesita liberar recursos para la nueva actividad.

- **Parada o detenida:** Ha pasado a segundo plano y está completamente tapada por la nueva actividad, en ese caso el sistema también puede optar por cerrarla si necesita liberar recursos.
- **Destruída:** Ya no está disponible, se han liberado todos sus recursos y en caso de ser llamada, necesitaría comenzar un nuevo ciclo de vida.

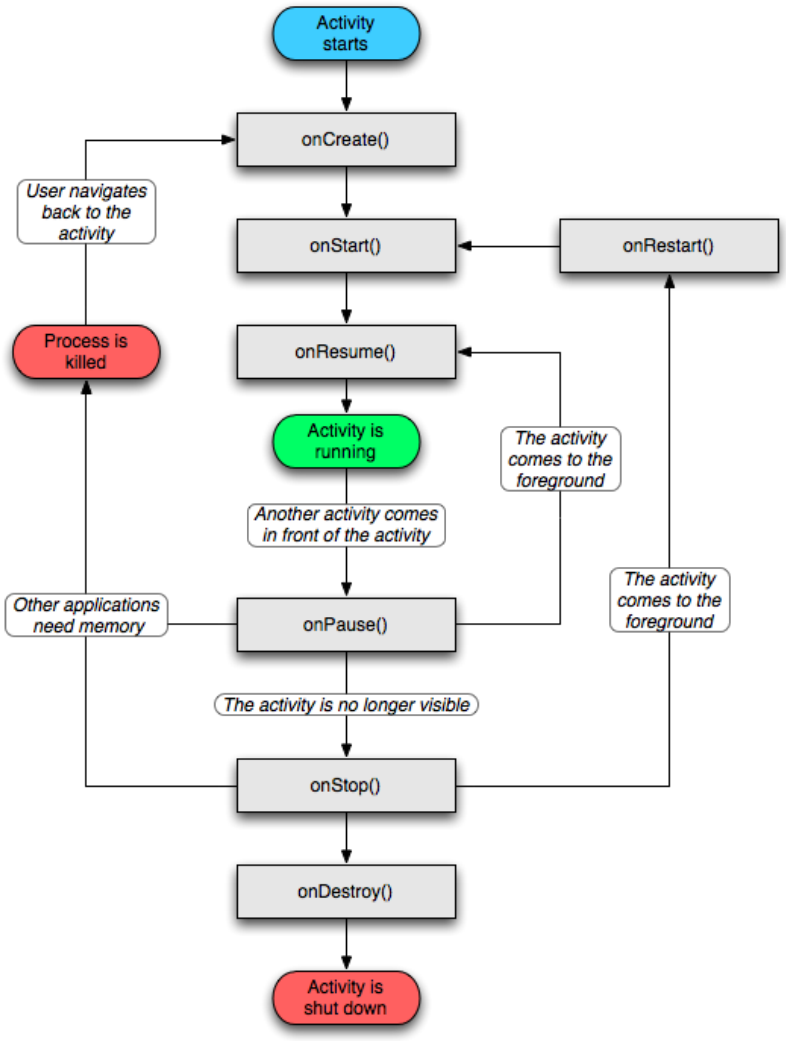

Figura 2.23: Ciclo de vida de una *activity*

Services Un servicio es un componente que se ejecuta en *background* para llevar a cabo operaciones de larga duración o para efectuar tareas para procesos remotos. Un servicio no proporciona una interfaz de usuario. Por ejemplo, un servicio es reproducir música en *background* mientas el usuario utiliza una

aplicación distinta, o reunir datos a través de la red sin bloquear la interacción del usuario con la *activity*.

Content Providers Un *content provider* administra un conjunto de datos que pueden ser compartidos entre las aplicaciones. Una aplicación puede almacenar datos en el sistema, en una base de datos SQLite, en la web, o en cualquier otra localización persistente a la que pueda acceder. A través de un *content provider* otras aplicaciones pueden acceder o modificar estos datos, siempre que el *content provider* lo permita. Por ejemplo, Android proporciona un *content provider* que administra la información de los contactos del usuario. De esta manera, una aplicación con los permisos adecuados puede consultar o modificar la información almacenada sobre los contactos.

Broadcast Receivers Un *broadcast receiver* es un componente que responde a anuncios globales del sistema. Muchos de estos anuncios son generados por el propio sistema por ejemplo, distintos *broadcasts* anunciando que la pantalla se ha apagado, el nivel de la batería es bajo o se ha tomado una foto. No obstante, las aplicaciones pueden generar sus propios *broadcasts* para avisar al resto de aplicaciones de algún evento. Los *broadcast receiver* no muestran una interfaz de usuario pero pueden crear un notificación en la barra de estado para indicar cuando ha ocurrido un evento.

Tres de los cuatro componentes previamente descritos (*activities*, *services* y *broadcast receivers*) son activados por un tipo de mensaje asíncrono llamado *intent*. Los *intents* permiten el paso de información en tiempo de ejecución entre estos componentes, ya sean de la misma aplicación o de aplicaciones distintas, a través de la cola de eventos de Android.

El manifiesto (AndroidManifest.xml)

Antes de que Android pueda iniciar un componente de una aplicación, el sistema debe conocer la existencia de dicho componente a través del archivo AndroidManifest.xml de la aplicación. Una aplicación debe tener declarados todos sus componentes en este fichero que está situado en la carpeta raíz del proyecto de la aplicación.

El manifiesto es un archivo en formato XML en el que se definen las características generales de una aplicación Android:

- **Paquete:** Cadena que identifica de forma unívoca una aplicación. No es posible añadir una aplicación al Play Store de Android si ya existe otra aplicación con el mismo nombre de paquete. Del mismo modo, si se instala en un dispositivo una aplicación con el mismo nombre de paquete que otra ya instalada, la nueva sustituirá a la anterior.
- **Nombre:** Nombre de la aplicación que permite que los usuarios la identifiquen.
- **Versión:** Declara la versión mínima de Android que necesita un dispositivo para poder instalar la aplicación.

- **Permisos:** Lista de permisos necesarios para que la aplicación se ejecute correctamente. Esta lista se le presentará al usuario cuando instale la aplicación.
- **Componentes:** Lista de componentes que forman la aplicación.

2.2.6. Seguridad en Android

Como se mencionó anteriormente, Android está basado en la versión 2.6 del kernel de Linux que es el que proporciona los servicios de seguridad. A lo largo de su historia, el kernel de Linux ha sido investigado, atacado y reparado por miles de desarrolladores convirtiéndose en un kernel estable y seguro en el que confían grandes empresas y profesionales de la seguridad llegando a ser usado en millones de entornos sensibles a la seguridad. En el entorno de la computación en sistemas móviles, el kernel de Linux proporciona a Android varias características clave en la seguridad del sistema operativo:

- Modelo de separación de privilegios basado en usuarios
- Aislamiento de procesos
- Mecanismo extensible para una comunicación entre procesos segura
- Habilidad para eliminar partes innecesarias y potencialmente inseguras del kernel

Como sistema operativo multiusuario, uno de los principales objetivos del kernel de Linux es aislar los recursos de los usuario. De manera que:

- El usuario A no pueda leer los archivos del usuario B
- El usuario A no consuma toda la memoria y deje sin nada al usuario B
- El usuario A no consuma todos los recursos de la CPU y deje sin nada al usuario B
- El usuario A no consuma todos los dispositivos (telefonía, GPS, Bluetooth...) y deje sin nada al usuario B

Además, todas las aplicaciones que se ejecutan en Android están sujetas a restricciones de seguridad impuestas por el framework de aplicaciones. A continuación se muestran algunos de los aspectos más importantes de la seguridad en Android:

Separación de privilegios El kernel de Android implementa un modelo de separación de privilegios como medio para identificar y aislar los recursos de las aplicaciones. De esta manera, todas las aplicaciones del sistema operativo Android se ejecutan en su propia instancia de la máquina virtual *Dalvik* con su propio identificador de usuario (uid) e identificador de grupo (gid). Esto evita que aplicaciones o procesos sin permisos puedan acceder a otras aplicaciones o procesos proporcionando un *Application Sandbox* a nivel de kernel. Por defecto, las aplicaciones no pueden interactuar entre si y tienen acceso limitado al sistema operativo.

Debido a que la puesta en marcha de la máquinas virtuales individuales puede incrementar notablemente la latencia entre el inicio de una aplicación y su ejecución, Android utiliza un mecanismo de precarga que permite acelerar el procedimiento. Para ello se utiliza un proceso que recibe el nombre de *Zygote* y que tiene dos funciones: primero, actúa como una plataforma de lanzamiento para nuevas aplicaciones y, en segundo lugar, actúa como un repositorio de bibliotecas al que pueden referirse la aplicaciones durante su ciclo de vida. El proceso *Zygote* se encarga de poner en marcha cada instancia de máquina virtual, la precarga y preinicializa con las bibliotecas básicas requeridas. A continuación, se mantiene la espera de recibir una señal para iniciar la aplicación. *Zygote* se inicia al arrancar el sistema y funciona de manera similar a una cola. Cualquier dispositivo Android tendrá siempre un proceso principal *Zygote* funcionando.

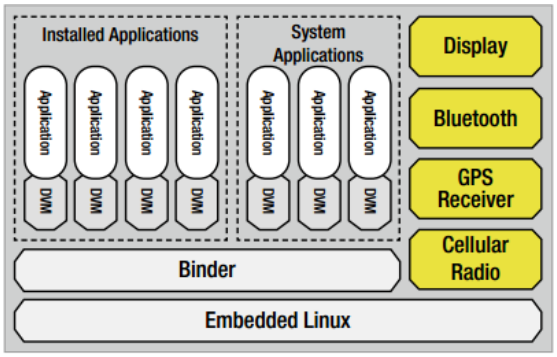

Figura 2.24: Arquitectura de seguridad Android

En algunos sistemas operativos, los errores de corrupción de memoria comprometen generalmente la seguridad del dispositivo. Esto no ocurre en Android, debido a que todas las aplicaciones y sus recursos están aislados a nivel de sistema operativo. Un error de corrupción de memoria únicamente permitiría la ejecución de código arbitrario en el contexto de la aplicación atacada.

Como todas las características de seguridad, el aislamiento de aplicaciones no es irrompible. Sin embargo, para romper el *Application Sandbox* en un dispositivo configurado correctamente, es necesario comprometer la seguridad del kernel de Linux.

Permisos Por defecto, una aplicación de Android solo puede acceder a una serie de recursos limitados del sistema. El sistema gestiona el acceso a los recursos de las aplicación Android de modo que, si se utilizan incorrecta o maliciosamente, pueden afectar desfavorablemente a la experiencia del usuario, la red, o los datos en el dispositivo.

Estas restricciones se aplican de formas diferentes. Algunas funciones están restringidas por una falta intencional de API de la funcionalidad (por ejemplo, no hay ninguna API de Android para la manipulación directa de la tarjeta SIM). En algunos casos, la separación de privilegios proporciona una medida de seguridad, como ocurre con el aislamiento de almacenamiento para cada aplicación. En otros casos, las APIs serán usadas por aplicaciones de confianza

y protegidas a través de un mecanismo de seguridad conocido como permisos. Estas APIs protegidas incluyen:

- Funciones de la cámara
- GPS
- Funciones Bluetooth
- Funciones de telefonía
- Funciones SMS/MMS
- Red/Conexión de datos

Estos recursos son solamente accesibles a través del sistema operativo. Para poder hacer uso de estas APIs en el dispositivo una aplicación debe definir las funcionalidades que necesita en su manifiesto (AndroidManifest.xml). Cuando se instala una aplicación, el sistema muestra un cuadro de diálogo al usuario que indica los permisos que necesita y le pregunta si desea continuar con la instalación. Si el usuario continúa con la instalación, el sistema acepta que el usuario ha concedido todos los permisos solicitados. El usuario no puede conceder o denegar permisos individuales, el usuario debe conceder o denegar todos los permisos solicitados como un bloque.

Dentro de los ajustes del dispositivo, los usuarios pueden ver los permisos que tienen las aplicaciones que han instalado previamente. Los usuarios también pueden desactivar algunas funciones a nivel global cuando se elige, por ejemplo, deshabilitar GPS, radio o Wi-Fi. En caso de que una aplicación intente utilizar una funcionalidad protegida que no se ha declarado en el manifiesto de la aplicación, la falta de permiso típicamente resultará en una excepción de seguridad será devuelta a la aplicación.

Firma del código de las aplicaciones La firma del código de las aplicaciones permite a los desarrolladores identificar al autor de la aplicación y actualizar las aplicaciones sin necesidad de crear un sistema complejo de interfaces y permisos. Todas las aplicaciones que se ejecutan en la plataforma Android tienen que haber sido firmadas por el desarrollador. Cualquier aplicación sin firmar será rechazada, ya sea por Google Play o el instalador de paquetes en el dispositivo Android.

En Google Play, la firma de aplicaciones establece un vínculo de confianza entre Google y el desarrollador, y entre el desarrollador y su aplicación. Las aplicaciones son distribuidas a través de Google Play sin ser modificadas, por lo que los propios desarrolladores son considerados responsables del comportamiento de las mismas.

En Android, la firma de aplicaciones es el primer paso para el aislamiento de procesos ya que el certificado firmado de la aplicación define el identificador asociado a la aplicación. La firma de aplicaciones evita que una aplicación no pueda acceder a otra si no tiene permisos.

Cuando una aplicación (archivo APK) se instala en un dispositivo Android, el administrador de paquetes verifica que la APK ha sido firmada correctamente con el certificado que incluye. Las aplicaciones pueden ser autofirmadas

o firmadas por una tercera parte (Original Equipment Manufacturer - OEM, operador...). Android permite la firma de aplicaciones utilizando certificados autofirmados que los desarrolladores pueden generar sin asistencia o permisos externos.

2.2.7. NFC en Android

El soporte para Near Field Communication en Android comienza a partir de la versión 2.3. La API inicial únicamente permitía la lectura de etiquetas. Esta API no permitía a los desarrolladores aprovechar el potencial de NFC por lo que en la siguiente versión se incluyeron nuevas características. Android 2.3.3 proporciona un soporte mejorado y extendido para NFC que permite a las aplicaciones interactuar con más tipos de etiquetas de distintas maneras. La nueva API permitía la lectura y escritura en una amplia gama de estándares para etiquetas como:

- NFC-A (ISO 14443-3A)
- NFC-B (ISO 14443-3B)
- NFC-F (JIS 6319-4)
- NFC-V (ISO 15693)
- ISO-DEP (ISO 14443-4)
- MIFARE Classic
- MIFARE Ultralight
- NFC Forum NDEF tags

La plataforma también proporciona un protocolo de comunicación peer-to-peer limitado y su correspondiente API. Además, es posible formatear distintos tipos de mensajes NDEF (texto, URL, MIME...).

A partir de la versión de Android 4.0 se incluyó la nueva funcionalidad Android Beam. Android Beam permite enviar mensajes NDEF entre dos dispositivos Android. Esta funcionalidad sustituye al peer-to-peer incluido en APIs anteriores. El protocolo utilizado para establecer la comunicación peer-to-peer puede ser NDEF Push Protocol - NPP (propio de Android) o Simple NDEF Exchange Protocol - SNEP (estándar del NFC Forum). El protocolo NPP es necesario para las versiones Android desde la 2.3.3 a la 3.2 pero desde la versión 4.0 se acepta tanto NPP como SNEP para establecer la comunicación peer-to-peer. La versión 4.0 también incluyó soporte para más tipos de etiquetas como:

- NDEF Formatable
- NFC Barcode

Dispositivos Android con NFC

En la actualidad, cada vez es más sencillo encontrar teléfonos móviles Android que incluyan Near Field Communication entre sus características. No obstante, el primer teléfono móvil que disponía de la tecnología NFC no apareció hasta 2006 y fue el Nokia 6131 (Figura 2.25).

Figura 2.25: Nokia 6131 NFC

Figura 2.26: Samsung Nexus S

Hasta 2009 no salió al mercado el primer dispositivo Android con funcionalidad NFC de la mano de Samsung, el Samsung Nexus S (Figura 2.26). La multinacional surcoreana ha apostado fuertemente por la tecnología Near Field Communication y actualmente la ha incluido en prácticamente todos los últimos modelos que han sacado a la venta como el Samsung Galaxy Nexus (Figura 2.27) o el Samsung Galaxy S III (Figura 2.28).

Figura 2.27: Samsung Galaxy Nexus

Figura 2.28: Samsung Galaxy S III

Los últimos lanzamientos de dispositivos Android por parte de Google también incluyen funcionalidad NFC tanto en la gama de tabletas como en el nuevo smartphone. La tableta Nexus 7 salió a la venta en julio de 2012 y fue el primer dispositivo de estas características en incluir NFC. En noviembre del mismo año año Google presentó a su hermana mayor, la tableta Nexus 10, y el nuevo smartphone Nexus 4, que esta vez estaría fabricado por LG. De nuevo una multinacional surcoreana, que ya estaba apostando también por el NFC en sus últimos modelos, será la encargada de fabricar un smartphone de Google.

Figura 2.29: Dispositivos Nexus de Google

2.3. Aplicaciones de transporte público en dispositivos móviles

2.3.1. Proyectos piloto

En febrero de 2012, Juniper Research publicó un estudio [14] sobre el crecimiento del *mobile ticketing* y se estimó que en el año 2016 se habrán vendido unos 23 billones de tiques para dispositivos móviles. El estudio indica que en 2016, la venta de tiques móviles NFC representará más del 50 % del total de los ingresos procedentes de la venta de tiques móviles. El volumen previsto sugiere que, con el tiempo, el *mobile ticketing* estará disponible para otros sectores además del sector del transporte.

Varios proyectos piloto han demostrado con éxito los beneficios potenciales del uso de NFC para la venta y uso de tiques asi como para el acceso a la información sobre horarios y líneas de transporte.

NJ TRANSIT, 2011 New Jersey Transit permitió a sus usuarios pagar sus tiques a través de un teléfono móvil con NFC equipado con *Google Wallet* en unas determinadas estaciones y en unas líneas de servicio concretas. *Google Wallet* es un sistema de pago móvil desarrollado por Google que permite a los usuarios almacenar tarjetas de crédito o débito, tarjetas de fidelidad, tarjetas regalo o descuentos entre otros. De esta manera, NJ Transit se convirtió en la primera agencia de tránsito de Estado Unidos en asociarse con *Google Wallet* para probar los pagos móviles permitiendo a sus clientes pagar con un simple toque de su teléfono móvil.

Londres, 2007 En el proyecto piloto con NFC en Londres hubo numerosos participantes entre los que se encontraban Transport for London (TfL), Transit/Cubic, el operador de telefonía móvil O2, Nokia, Barclaycard y Visa. Aproximadamente, unos 500 usuarios de O2 recibieron el teléfono con NFC Nokia 6131 con aplicaciones de la *Oyster Card* [15] y tarjetas de crédito disponibles. El teléfono permitía a los viajeros cargar una cantidad de dinero así como abonos mensuales en la *Oyster Card*. Los participantes podían utilizar el teléfono en los terminales disponibles para la *Oyster Card* en el metro, autobús y tranvía.

San Francisco, 2008 En 2008 se puso en marcha el *Bay Area Rapid Transit* (BART) que combinaba el uso de tiques de transporte con el pago móvil. Los usuarios de BART empleaban un teléfono Samsung con NFC para acceder a las estaciones en la zona de la bahía de San Francisco así como para pagar en los restaurantes *Jack in the Box*. Los participantes en el proyecto fueron ViVOtech, Cubic, Sprint y First Data. Durante el proyecto también se colocaron *smartposters* que incorporaban tiques de descuento que podían ser usados por los clientes.

Alemania, 2008 La autoridad ferroviaria alemana, Deutsche Bahn, junto con sus socios, Vodafone, Deutsche Telekom y O2 Alemania, llevaron a cabo un proyecto piloto con NFC en sus trenes interurbanos que conectaban las ciudades de Berlín, Colonia, Dusseldorf, y Frankfurt; los servicios ferroviarios locales en Berlín y los servicios de tránsito de Potsdam. El proyecto contó con 3000 participantes que usaban en los teléfonos una aplicación llamada «Touch&Travel». En la estación origen, los usuarios hacen *check-in* pasando su teléfono por el terminal de entrada y, al salir, se repite el proceso haciendo *check-out* en un terminal de la estación destino; de esta manera se realizará el cálculo de la tarifa que debe pagar el viajero.

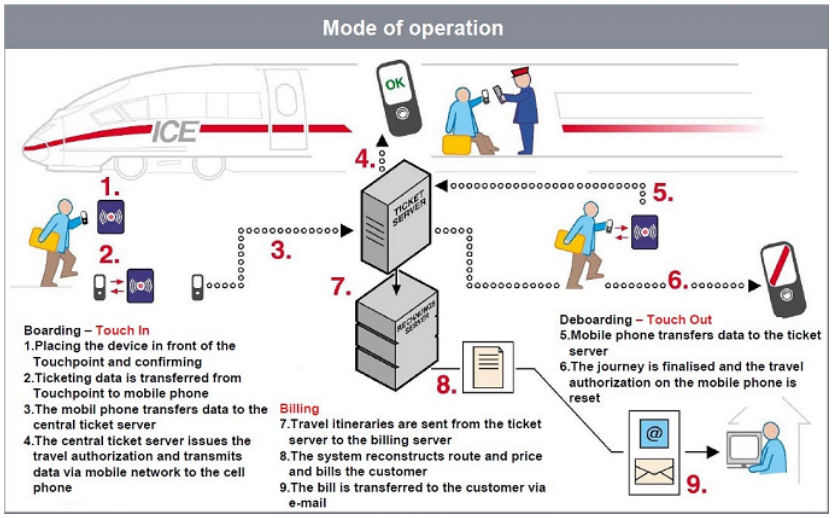

Figura 2.30: Visión general del sistema Touch&Travel

Málaga, 2011 Gracias a la iniciativa puesta en marcha por Orange y la Empresa Malagueña de Transportes (EMT), los usuarios del autobús urbano de Málaga pudieron llevar su Tarjeta Transbordo en un móvil NFC de la operadores y disponer de información en tiempo real. Esta iniciativa contó con 150 usuarios a los que se proporcionó un teléfono móvil NFC con la aplicación Cartera Orange. A través de ella, los usuarios podían comprar su título de transporte y validar cada viaje en el momento de subir al autobús sólo con acercar el teléfono al lector situado dentro del vehículo. Asimismo, se podía consultar el saldo de viajes restante en su título de transporte y consultar el histórico de los viajes realizados.

2.3.2. Proyectos en funcionamiento

En la actualidad es difícil encontrar sistemas ya implantados que permitan utilizar el móvil como tique haciendo uso de NFC más allá de proyectos piloto. A continuación se muestran un par de ejemplos de aplicaciones de *mobile ticketing* que no hacen uso de NFC para transmitir el tique.

ARRIVA m-ticket Desde 2010 la compañía multinacional de transporte público propiedad de Deutsche Bahn, Arriva, ofrece a sus usuarios de autobús de Reino Unido la posibilidad de adquirir los tiques a través de sus teléfonos móvil. Los usuarios se registran en la aplicación introduciendo sus datos así como el número de una tarjeta de crédito a la que se cargaran las compras realizadas. Una vez registrados, ya pueden acceder a la aplicación para comprar sus tiques. Los tiques se almacenan en el dispositivo y en el momento de uso el viajero activa el tique y se lo muestra al conductor del autobús para poder acceder.

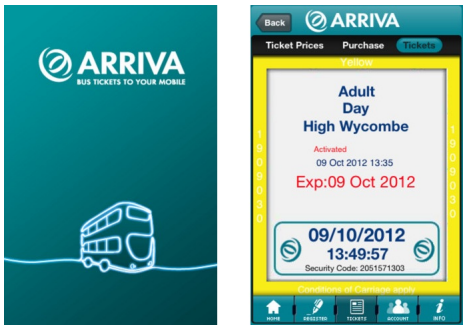

Figura 2.31: Aplicación de Arriva

RMV HandyTicket / RNV HandyTicket Los consorcios de transporte de las regiones alemanas de Rhein-Main-Verkehrsverbund (RMV) y Rhein-Neckar-Verkehr (RNV) ofrecen a sus clientes la posibilidad de comprar los tiques a través de su teléfono móvil. Al igual que ocurre con la aplicación de Arriva, los usuarios deben registrarse en sistema introduciendo sus datos así como el número de una tarjeta de crédito a la que se cargaran las compras realizadas. Una vez registrados, ya pueden acceder a la aplicación y comprar sus tiques. Los tiques se almacenan en el dispositivo y el usuario debe mostrarlos para poder acceder al medio de transporte.

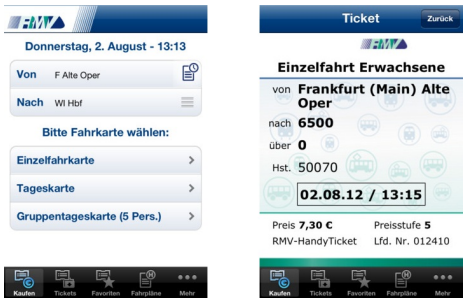

Figura 2.32: Aplicación de RMV

3

Análisis

3.1. Introducción

En este apartado se va a detallar el análisis del sistema a desarrollar siguiendo las pautas recogidas por la Agencia Espacial Europea en la versión lite del documento que especifica como aplicar sus estándares a proyectos software pequeños [16].

En primer lugar se realizará una descripción general del sistema que permitirá al lector tener una idea aproximada del sistema final que se va a desarrollar. A continuación se especificaran los requisitos de usuario que debe cumplir el sistema y se indicaran los casos de uso que indicarán que funcionalidades tiene el mismo. Por último, se recogerán los requisitos software que derivan de los requisitos de usuario y que serán la base del diseño del sistema.

3.2. Descripción General

El sistema que se va a desarrollar consiste en una aplicación para teléfonos móviles Android que permite la adquisición de tiques para transporte público para su posterior uso utilizando la tecnología Near Field Communication, permitiendo así agilizar el proceso tradicional.

El sistema completo constará de tres partes principales:

- **Aplicación móvil:** La aplicación móvil es la que el usuario descarga e instala en su dispositivo. La aplicación permite al usuario adquirir tiques, almacenarlos de manera segura en su dispositivo, usarlos a través de NFC y consultar la relación de tiques adquiridos tanto sin usar como usados.
- **Servidor:** El servidor aloja una base de datos que almacena toda la información relacionada con los tiques. La comunicación de la aplicación con el servidor se realizará a través de servicios web REST. El servidor se encargará de emitir tiques válidos que llevaran su firma digital siempre que un usuario lo solicite a través de la aplicación móvil.
- **Lector NFC:** El lector de NFC estará conectado a un equipo que permitirá realizar la lectura de los tiques enviados desde el teléfono móvil por

los usuarios. Antes de permitir el acceso, se realizará una validación del tique comprobando tanto que el tique no ha caducado como que la firma incluida es correcta.

3.3. Requisitos de Usuario

En el análisis de un proyecto software se realiza la definición de requisitos de usuario, que especifican a alto nivel tanto las capacidades del sistema como las restricciones que establecen cómo deben llevarse a cabo dichas funcionalidades. La plantilla que va a utilizarse para la especificación de los requisitos de usuario es la siguiente:

Identificador: RU<tipo>-<número>	
Título:	
Descripción:	
Prioridad:	
Necesidad:	
Fuente:	

Tabla 3.1: Plantilla de tabla de requisito de usuario

- **Identificador:** Código alfanumérico que identifica unívocamente al requisito. La sintaxis del identificador es RU<tipo>-<número> siendo RU un indicador que hace referencia a que es un requisito de usuario, <tipo> un carácter que indica el tipo de requisito de usuario (C para los requisitos de capacidad o R para los requisitos de restricción) y <número> una cifra de dos dígitos que se incrementa por cada requisito y lo identifica dentro los requisitos de su mismo tipo.

- **Título:** Cadena que resume la funcionalidad de un requisito. Es más sencillo de recordar que el identificador y más breve que la descripción del requisito.

- **Descripción:** Descripción textual breve del requisito, detallando su información de forma clara y concisa.

- **Prioridad:** Nivel de preferencia de la implementación del requisito durante la fase de desarrollo de la aplicación. Puede tomar los valores “Alta”, “Media” o “Baja”.

- **Necesidad:** Grado en que se puede prescindir o no de la implementación de un requisito. Puede tomar los valores “Esencial”, “Deseable” u “Opcional”.

- **Fuente:** Indica la persona o documento del cual ha sido extraído el requisito. Los requisitos pueden proceder de exigencias del cliente o de recomendaciones de expertos en el ámbito que abarca el proyecto.

3.3.1. Requisitos de Capacidad

Identificador: RUC-01	
Título:	Iniciar la aplicación móvil
Descripción:	El usuario podrá iniciar la aplicación móvil.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.2: Requisito de capacidad RUC-01

Identificador: RUC-02	
Título:	Adquirir tique
Descripción:	El usuario puede adquirir tiques de transporte público a través de la aplicación móvil.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.3: Requisito de capacidad RUC-02

Identificador: RUC-03	
Título:	Mostrar tiques adquiridos sin usar
Descripción:	El usuario puede consultar los tiques que ha adquirido a través de la aplicación.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.4: Requisito de capacidad RUC-03

Identificador: RUC-04	
Título:	Mostrar tiques adquiridos usados el día actual
Descripción:	El usuario puede consultar los tiques que ha adquirido y usado el día actual a través de la aplicación.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.5: Requisito de capacidad RUC-04

Identificador: RUC-05	
Título:	Establecer código de seguridad
Descripción:	Al acceder a la aplicación el usuario deberá establecer un código de seguridad que la protegerá de un uso indebido por parte de usuarios no autorizados.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.6: Requisito de capacidad RUC-05

Identificador: RUC-06	
Título:	Usar un tique para acceder del servicio de transporte público
Descripción:	El usuario puede usar los tiques adquiridos a través de la aplicación para acceder al servicio de transporte público indicado en el tique.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.7: Requisito de capacidad RUC-06

Identificador: RUC-07	
Título:	Usar un tique para salir del servicio de transporte público
Descripción:	El usuario puede enviar el tique usado para acceder ese día al lector situado en la salida del servicio de transporte público.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.8: Requisito de capacidad RUC-07

Identificador: RUC-08	
Título:	Enviar un tique al revisor del servicio de transporte público
Descripción:	El usuario puede enviar el tique usado para acceder ese día al servicio de transporte público al lector del revisor si se diera el caso.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.9: Requisito de capacidad RUC-08

Identificador: RUC-09	
Título:	Refrescar tiques
Descripción:	El usuario puede refrescar los tiques adquiridos en cualquier momento desde la aplicación móvil.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.10: Requisito de capacidad RUC-09

Identificador: RUC-10	
Título:	Cambiar código de seguridad
Descripción:	El usuario puede cambiar el código se seguridad en cualquier momento desde la aplicación móvil.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.11: Requisito de capacidad RUC-10

Identificador: RUC-11	
Título:	Consultar todos los tiques adquiridos
Descripción:	La aplicación móvil permitirá al usuario consultar todos los tiques adquiridos.
Prioridad:	Media
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.12: Requisito de capacidad RUC-11

Identificador: RUC-12	
Título:	Consultar si el usuario está bloqueado
Descripción:	La aplicación móvil permitirá al usuario consultar si la aplicación le ha bloqueado.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.13: Requisito de capacidad RUC-12

Identificador: RUC-13	
Título:	Salir de la aplicación móvil
Descripción:	El usuario puede salir de la aplicación móvil en cualquier momento sin perder los tiques adquiridos.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.14: Requisito de capacidad RUC-13

3.3.2. Requisitos de Restricción

Identificador: RUR-01	
Título:	Uso del sistema sin registro de usuarios
Descripción:	El sistema no dispone de registro de usuarios por lo que un usuario no tiene que registrarse en el sistema para poder utilizarlo.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.15: Requisito de restricción RUR-01

Identificador: RUR-02	
Título:	Idioma de la aplicación móvil
Descripción:	La aplicación móvil estará disponible tanto en castellano como en inglés.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.16: Requisito de restricción RUR-02

Identificador: RUR-03	
Título:	Tiempos de respuesta
Descripción:	El sistema debe responder en el menor tiempo posible a las peticiones realizadas por la aplicación móvil.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.17: Requisito de restricción RUR-03

Identificador: RUR-04	
Título:	Sistema operativo del dispositivo móvil
Descripción:	La aplicación móvil funcionará únicamente en dispositivos con el sistema operativo Android versión 4.1.2 o superior.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.18: Requisito de restricción RUR-04

Identificador: RUR-05	
Título:	Conectividad del dispositivo móvil
Descripción:	La aplicación móvil necesitará que el dispositivo tenga conectividad a Internet para llevar a cabo operaciones que requieran conexión con el servidor.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.19: Requisito de restricción RUR-05

Identificador: RUR-06	
Título:	Conectividad durante el envío de tiques
Descripción:	La aplicación móvil no necesitará que el dispositivo disponga de conectividad a Internet para enviar los tiques al lector.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.20: Requisito de restricción RUR-06

Identificador: RUR-07	
Título:	Disponibilidad de NFC
Descripción:	El dispositivo móvil deberá disponer de chip NFC para el correcto funcionamiento de la aplicación.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.21: Requisito de restricción RUR-07

Identificador: RUR-08	
Título:	Envío de tiques a través de NFC
Descripción:	Los tiques tanto sin usar como usados se enviarán a través de la tecnología NFC.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.22: Requisito de restricción RUR-08

Identificador: RUR-09	
Título:	Lector NFC
Descripción:	El lector NFC que va a utilizar el sistema es el SCL3711 de SCM Microsystem.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.23: Requisito de restricción RUR-09

Identificador: RUR-10	
Título:	Comunicación con el servidor a través de servicios web REST
Descripción:	La comunicación con el servidor se realiza a través de servicios web REST.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.24: Requisito de restricción RUC-10

Identificador: RUR-11	
Título:	Comunicación segura con el servidor
Descripción:	La comunicación con los servicios web del servidor se hará de manera segura utilizando el protocolo HTTPS.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.25: Requisito de restricción RUC-11

Identificador: RUR-12	
Título:	Vibración de envío de tique NFC
Descripción:	El dispositivo móvil vibrará cuando se envíe un tique a través de NFC.
Prioridad:	Media
Necesidad:	Deseable
Fuente:	Cliente

Tabla 3.26: Requisito de restricción RUC-12

Identificador: RUR-13	
Título:	Almacenamiento de tiques en el teléfono móvil
Descripción:	La aplicación debe almacenar en el dispositivo móvil los tiques adquiridos por el usuario para permitir el uso *offline* de la misma.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.27: Requisito de restricción RUR-13

Identificador: RUR-14	
Título:	Almacenamiento seguro de tiques en el teléfono móvil
Descripción:	Los tiques adquiridos por el usuario se almacenaran en el dispositivo móvil de forma segura evitando así un posible uso fraudulento de los mismos.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.28: Requisito de restricción RUR-14

Identificador: RUR-15	
Título:	Información visual
Descripción:	La aplicación móvil deberá informar al usuario de manera visual del resultado de una acción.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.29: Requisito de restricción RUR-15

Identificador: RUR-16	
Título:	Petición de código de seguridad código de seguridad
Descripción:	La aplicación móvil deberá solicitar el código de seguridad antes de realizar determinadas operaciones como adquirir tiques, usar tiques, refrescar los tiques, cambiar el código de seguridad y comprobar si el usuario está en la lista negra del sistema.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.30: Requisito de restricción RUR-16

Identificador: RUR-17	
Título:	Bloqueo de usuarios
Descripción:	El sistema podrá bloquear la aplicación móvil a los usuarios que intenten hacer un uso fraudulento (falsificación de tiques, tiques duplicados...) o introduzcan mal el código de seguridad un número determinado de veces.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.31: Requisito de restricción RUR-17

Identificador: RUR-18	
Título:	Registro de eventos
Descripción:	El sistema registrará cada acción que realice un usuario desde la aplicación móvil con el servidor.
Prioridad:	Media
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.32: Requisito de restricción RUR-18

Identificador: RUR-19	
Título:	Almacenamiento de tiques en el servidor
Descripción:	El servidor almacenará de manera persistente los tiques adquiridos por todos los usuarios.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	Cliente

Tabla 3.33: Requisito de restricción RUR-19

Identificador: RUR-20	
Título:	Acceso remoto al servidor
Descripción:	Se podrá acceder de manera remota al servidor.
Prioridad:	Media
Necesidad:	Deseable
Fuente:	Cliente

Tabla 3.34: Requisito de restricción RUR-20

3.4. Casos de Uso

Los casos de uso nacen de los requisitos siendo su representación gráfica. El objetivo de este apartado es modelar y definir textualmente los casos de uso de la aplicación, que describen las funcionalidades principales de la misma y la interacción de los usuarios en diferentes escenarios. Como sólo se dispone de un rol o tipo de actor interactuando con la aplicación, todos los casos de uso describirán las interacciones que puede llevar a cabo cualquier usuario así como las respuestas obtenidas.

Cada caso de uso describe los pasos que debe seguir el actor (en este caso, el usuario) para lograr utilizar una determinada funcionalidad. Los casos de uso se muestran en el siguiente diagrama:

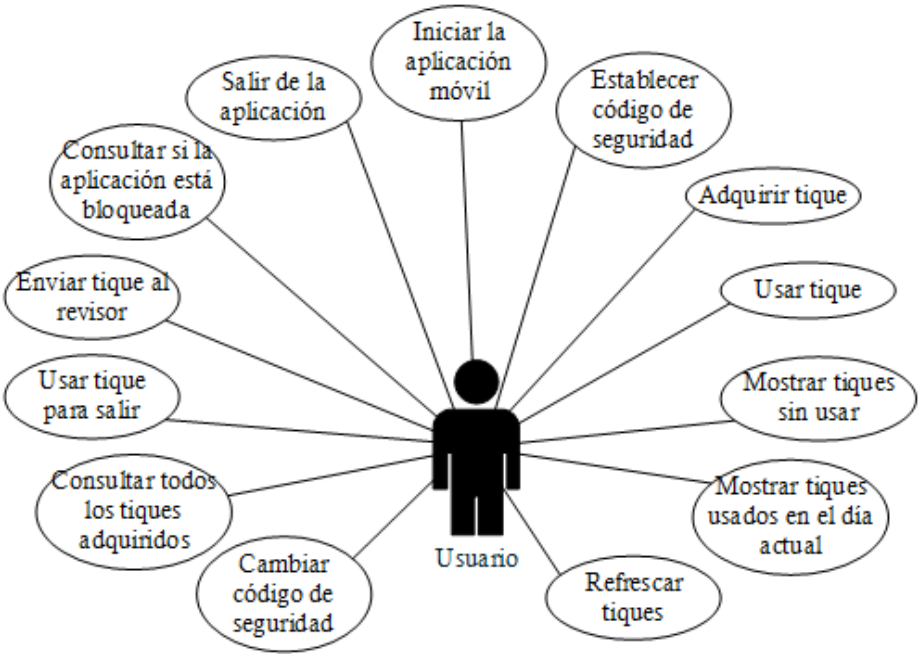

Figura 3.1: Diagrama de casos de uso

Seguidamente, se van a describir textualmente cada uno de los casos de uso de la aplicación. Para ello, se usará la plantilla que se muestra a continuación:

Identificador: CU-<número>	
Nombre:	
Objetivo:	
Precondiciones:	
Postcondiciones:	
Escenario básico:	
Escenario alternativo:	

Tabla 3.35: Plantilla de tabla de caso de uso

- **Identificador:** Código alfanumérico que identifica unívocamente al requisito. La sintaxis del identificador es CU-<número> siendo CU un indicador que hace referencia a que es un caso de uso y <número> una cifra de dos dígitos que se incrementa por cada caso de uso.

- **Nombre:** Nombre del caso de uso, que resume la funcionalidad referida por el mismo.

- **Objetivo:** Descripción textual breve del caso de uso.

- **Precondiciones:** Listado de todas las condiciones que deben cumplirse y acciones que deben haber tenido lugar antes del inicio del caso de uso.

- **Postcondiciones:** Estado del sistema tras ejecutarse el caso de uso.

- **Escenario básico:** Enumeración ordenada de las acciones del usuario y respuestas del sistema que tendrá lugar durante la ejecución del caso de uso en condiciones normales. Esto es, son los pasos deben seguirse para ejecutar la funcionalidad del caso de uso.

- **Escenario alternativo:** Bifurcación en la secuencia de pasos descrita por el escenario básico que describe los casos en los que la funcionalidad esperada no puede llevarse a cabo por unas determinadas causas.

Identificador: CU-01	
Nombre:	Iniciar la aplicación móvil
Objetivo:	El usuario inicia la aplicación.
Precondiciones:	El teléfono móvil deberá estar encendido.
Postcondiciones:	La aplicación se inicia correctamente.
Escenario básico:	1. El usuario entra al menú de su teléfono móvil Android. 2. El usuario selecciona la aplicación. 3. Durante el inicio de la sesión se realiza una comprobación para verificar que el usuario no aparece en la lista negra y se actualizan, si fuera necesario, la información de los tiques usados. 4. La comprobación se realiza correctamente y la aplicación se inicia con normalidad.
Escenario alternativo:	4. Si el dispositivo no dispone de conectividad o no es posible establecer comunicación con el servidor, la aplicación muestra un mensaje indicando que no ha sido posible comunicarse con el servidor y se inicia. 4. Si usuario se encuentra bloqueado por el sistema se mostrará un mensaje y la aplicación se cerrará.

Tabla 3.36: Caso de uso CU-01

Identificador: CU-02	
Nombre:	Establecer código de seguridad
Objetivo:	Al iniciar la aplicación móvil por primera vez se pedirá que el usuario establezca un código de seguridad.
Precondiciones:	Iniciar por primera vez la aplicación móvil.
Postcondiciones:	El código ha sido correctamente establecido y la aplicación se inicia con normalidad.
Escenario básico:	1. El usuario inicia por primera vez desde su instalación la aplicación móvil. 2. El usuario introduce un código de 4 dígitos en la aplicación. 3. La aplicación solicita al usuario que vuelva a introducir el código anterior. 4. Si ambos códigos introducidos por el usuario coinciden, el código queda establecido y la aplicación se inicia con normalidad.
Escenario alternativo:	5. Si los códigos introducidos no coinciden se repite todo el proceso desde el principio.

Tabla 3.37: Caso de uso CU-02

Identificador: CU-03	
Nombre:	Adquirir tique
Objetivo:	Adquirir un tique de transporte público a través de la aplicación.
Precondiciones:	Iniciar la aplicación móvil.
Postcondiciones:	El tique se adquiere con normalidad y se almacena en el dispositivo.
Escenario básico:	1. El usuario inicia la aplicación móvil. 2. El usuario selecciona la pestaña «Comprar Tickets» en la aplicación. 3. El usuario selecciona el tipo de transporte para el que desea adquirir el tique, el tipo de tique y la cantidad de tiques. 4. El usuario pulsa el botón comprar. 5. El dispositivo muestra el resumen de lo que el usuario desea adquirir y el coste total. 6. Si el usuario está de acuerdo, confirma la adquisición pulsando el botón «Aceptar». 7. La aplicación solicitará el código de seguridad al usuario para continuar con la operación. 8. Si el código introducido es correcto, la aplicación se comunicará con el servidor para proceder con la operación. 9. Si no se produce ningún error durante la operación, la aplicación recibirá el/los tique/s adquiridos y los almacenará/n en el dispositivo para su posterior uso.
Escenario alternativo:	5. Si el dispositivo no dispone de conectividad o no es posible establecer comunicación con el servidor, la aplicación muestra un mensaje indicando que no ha sido posible comunicarse con el servidor. 5. Si usuario se encuentra bloqueado por el sistema se mostrará un mensaje y no podrá continuar con la adquisición. 6. Si el usuario no desea continuar con la adquisición, pulsa el botón «Cancelar». 8. Si el código introducido es incorrecto, se cancelará la operación. 9. Si se produce un error durante la operación, se cancelará la operación.

Tabla 3.38: Caso de uso CU-03

Identificador: CU-04	
Nombre:	Usar tique
Objetivo:	Usar un tique para acceder a un servicio de transporte público.
Precondiciones:	Iniciar la aplicación móvil.
Postcondiciones:	El tique se usa con normalidad y el usuario accede al servicio de transporte público.
Escenario básico:	1. El usuario inicia la aplicación móvil. 2. El usuario selecciona la pestaña «Mis Tickets» en la aplicación. 3. El usuario selecciona el tique que desea utilizar. 4. La aplicación solicitará el código de seguridad al usuario para continuar con la operación. 5. Si el código introducido es correcto, la aplicación muestra la información del tique seleccionado y lo prepara para ser enviado al lector NFC. 6. El usuario coloca su dispositivo sobre el lector NFC. 7. Si el tique es válido, el usuario accede al servicio de transporte público.
Escenario alternativo:	4. Si usuario se encuentra bloqueado por el sistema se mostrará un mensaje y no podrá continuar con la operación. 5. Si el código introducido es incorrecto, se cancela la operación. 7. Si el tique no es válido, el usuario no puede acceder al servicio de transporte público. 8. Si un usuario envía 3 veces un tique no válido o ya usado, el sistema considera que el usuario está intentado hacer un uso fraudulento del mismo y automáticamente incluye al usuario en la lista negra.

Tabla 3.39: Caso de uso CU-04

Identificador: CU-05	
Nombre:	Mostrar tiques sin usar
Objetivo:	Mostrar los tiques adquiridos sin usar y que no están caducados.
Precondiciones:	Iniciar la aplicación móvil.
Postcondiciones:	La aplicación muestra todos los tiques adquiridos sin usar que no están caducados.
Escenario básico:	1. El usuario inicia la aplicación móvil. 2. El usuario selecciona la pestaña «Mis Tickets» en la aplicación.
Escenario alternativo:	No aplica.

Tabla 3.40: Caso de uso CU-05

Identificador: CU-06	
Nombre:	Mostrar tiques usados en el día actual
Objetivo:	Mostrar los tiques usados hoy.
Precondiciones:	Iniciar la aplicación móvil.
Postcondiciones:	La aplicación muestra todos los tiques adquiridos usados hoy.
Escenario básico:	1. El usuario inicia la aplicación móvil. 2. El usuario selecciona la pestaña «Tickets Usados» en la aplicación.
Escenario alternativo:	No aplica.

Tabla 3.41: Caso de uso CU-06

Identificador: CU-07	
Nombre:	Refrescar tiques
Objetivo:	Refrescar los tiques del dispositivo.
Precondiciones:	Iniciar la aplicación móvil.
Postcondiciones:	La aplicación refresca los tiques del dispositivo.
Escenario básico:	1. El usuario inicia la aplicación móvil. 2. El usuario pulsa el botón de opciones en la aplicación. 3. El usuario selecciona la opción «Refrescar» en el menú. 4. Los tiques del dispositivo se refrescan correctamente.
Escenario alternativo:	4. Si el dispositivo no dispone de conectividad o no es posible establecer comunicación con el servidor, la aplicación muestra un mensaje indicando que no ha sido posible comunicarse con el servidor. 4. Si usuario se encuentra bloqueado por el sistema se mostrará un mensaje y no podrá continuar con la operación.

Tabla 3.42: Caso de uso CU-07

Identificador: CU-08	
Nombre:	Cambiar código de seguridad
Objetivo:	Cambiar el código de seguridad de la aplicación.
Precondiciones:	Iniciar la aplicación móvil.
Postcondiciones:	El usuario cambia el código de seguridad de la aplicación correctamente.
Escenario básico:	1. El usuario inicia la aplicación móvil. 2. El usuario pulsa el botón de opciones en la aplicación. 3. El usuario selecciona la opción «Cambiar contraseña» en el menú. 4. La aplicación solicitará el código de seguridad al usuario para continuar con la operación. 5. Si el código es correcto, se procederá con el cambio de código. 6. El usuario introduce un código de 4 dígitos en la aplicación. 7. La aplicación solicita al usuario que vuelva a introducir el código anterior. 8. Si ambos códigos introducidos por el usuario coinciden, el código queda establecido y el nuevo código queda establecido.
Escenario alternativo:	5. Si el código introducido es incorrecto, se cancela la operación. 8. Si los códigos introducidos no coinciden el código no se cambia.

Tabla 3.43: Caso de uso CU-08

Identificador: CU-09	
Nombre:	Consultar todos los tiques adquiridos
Objetivo:	Consultar todos los tiques adquiridos (usados y sin usar) por el usuario.
Precondiciones:	Iniciar la aplicación móvil.
Postcondiciones:	Ver listado de tiques adquiridos.
Escenario básico:	1. El usuario inicia la aplicación móvil. 2. El usuario pulsa el botón de opciones en la aplicación. 3. El usuario selecciona la opción «Registro de tickets» en el menú. 4. Se muestra un listado con los tiques adquiridos.
Escenario alternativo:	3. Si el dispositivo no dispone de conectividad o no es posible establecer comunicación con el servidor, la aplicación muestra un mensaje indicando que no ha sido posible comunicarse con el servidor.

Tabla 3.44: Caso de uso CU-09

Identificador: CU-10	
Nombre:	Usar tique para salir
Objetivo:	Enviar tique para salir del servicio de transporte público.
Precondiciones:	Iniciar la aplicación móvil.
Postcondiciones:	El tique para salir se envía correctamente y el usuario sale del servicio de transporte público.
Escenario básico:	1. El usuario inicia la aplicación móvil. 2. El usuario selecciona la pestaña «Tickets Usados» en la aplicación. 3. El usuario seleccionado el tique usado previamente para acceder al servicio de transporte público. 4. La aplicación solicitará el código al usuario para continuar con la operación. 5. El usuario introduce el código para salir del servicio de transporte público indicado junto al lector. 6. La aplicación muestra la información del tique seleccionado y lo prepara para ser enviado al lector NFC. 7. El usuario coloca su dispositivo sobre el lector NFC. 8. Si el tique es válido, el usuario puede abandonar el servicio de transporte público.
Escenario alternativo:	8. Si el tique no es válido, el usuario no podrá abandonar con normalidad el servicio de transporte público. 9. Si un usuario envía 3 veces un tique no válido o ya usado para salir, el sistema considera que el usuario está intentado hacer un uso fraudulento del mismo y automáticamente incluye al usuario en la lista negra.

Tabla 3.45: Caso de uso CU-10

Identificador: CU-11	
Nombre:	Enviar tique al revisor
Objetivo:	Enviar tique al revisor del servicio de transporte público.
Precondiciones:	Iniciar la aplicación móvil.
Postcondiciones:	El tique se envía correctamente al revisor.
Escenario básico:	1. El usuario inicia la aplicación móvil. 2. El usuario selecciona la pestaña «Tickets Usados» en la aplicación. 3. El usuario seleccionado el tique usado previamente para acceder al servicio de transporte público. 4. La aplicación solicitará el código al usuario para continuar con la operación. 5. El usuario introduce el código facilitado por el revisor del servicio de transporte público. 6. La aplicación muestra la información del tique seleccionado y lo prepara para ser enviado al lector NFC. 7. El usuario coloca su dispositivo sobre el lector NFC. 8. Si el tique es válido, el revisor permite continuar el viaje del usuario.
Escenario alternativo:	8. Si el tique no es válido, el revisor tomará las medidas pertinentes.

Tabla 3.46: Caso de uso CU-11

Identificador: CU-12	
Nombre:	Consultar si la aplicación está bloqueada
Objetivo:	Comprobar si la aplicación está bloqueada debido a que el usuario ha hecho un uso fraudulento de la misma.
Precondiciones:	Iniciar la aplicación móvil.
Postcondiciones:	La aplicación muestra al usuario si está incluido en la lista negra del servicio.
Escenario básico:	1. El usuario inicia la aplicación móvil. 2. El usuario pulsa el botón de opciones en la aplicación. 3. El usuario selecciona la opción «Comprobar lista negra» en el menú. 4. La aplicación solicitará el código de seguridad al usuario para continuar con la operación. 5. El usuario introduce un código de 4 dígitos en la aplicación. 6. Si el código es correcto, se procederá con la consulta.
Escenario alternativo:	6. Si el código es incorrecto, se cancela la consulta.

Tabla 3.47: Caso de uso CU-12

Identificador: CU-13	
Nombre:	Salir de la aplicación
Objetivo:	Salir correctamente de la aplicación.
Precondiciones:	Iniciar la aplicación móvil.
Postcondiciones:	La aplicación se cierra correctamente.
Escenario básico:	1. El usuario inicia la aplicación móvil. 2. El usuario pulsa el botón atrás disponible en todos los dispositivos Android.
Escenario alternativo:	No aplica.

Tabla 3.48: Caso de uso CU-13

3.5. Requisitos Software

Los requisitos de software están orientados al programador y permiten definir de manera más técnica y detallada las características y capacidades de la aplicación a desarrollar.

En este apartado se van a detallar los requisitos software de la aplicación, tanto funcionales como no funcionales, obtenidos a través del análisis de los requisitos de usuario y de la información extraída del cliente. Los tipos de requisitos software que se van a especificar son los siguientes:

- **Requisito funcional (FU):** Define el «qué» debe hacer la aplicación.
- **Requisito de interfaz (IN):** Indica la interfaz que se usará para comunicarse con el usuario y con otros sistemas.
- **Requisito de operación (OP):** Indica «cómo» va a realizar el sistema las tareas para las que fue concebido.
- **Requisito de recursos (RE):** Especifican valores máximos de consumos de recursos por parte de nuestro sistema.
- **Requisito de comprobación(CO):** Indica «cómo» se debe de verificar los datos de entrada y de salida.
- **Requisito de seguridad (SE):** Indica los métodos de que dispone el sistema para asegurar la integridad, confidencialidad y disponibilidad de los datos.

A continuación se muestra, la plantilla que va a utilizarse para la especificación de los requisitos software:

Identificador: RU<tipo>-<número>	
Título:	
Descripción:	
Prioridad:	
Necesidad:	
Fuente:	

Tabla 3.49: Plantilla de tabla de requisito software

- **Identificador:** Código alfanumérico que identifica unívocamente al requisito. La sintaxis del identificador es RS<tipo>-<número> siendo RS un indicador que hace referencia a que es un requisito software, <tipo> un carácter que indica el tipo de requisito software (FU para los requisitos funcionales, IN para los requisitos de interfaz, OP para los requisitos de operación, RC para los requisitos de recursos, CO para los requisitos de comprobación y SE para los requisitos de seguridad) y <número> una cifra de dos dígitos que se incrementa por cada requisito y lo identifica dentro los requisitos de su mismo tipo.
- **Título:** Cadena que resume la funcionalidad de un requisito. Es más sencillo de recordar que el identificador y más breve que la descripción del requisito.

- **Descripción:** Descripción textual breve del requisito, detallando su información de forma clara y concisa.
- **Prioridad:** Nivel de preferencia de la implementación del requisito durante la fase de desarrollo de la aplicación. Puede tomar los valores "Alta", "Media" o "Baja".
- **Necesidad:** Grado en que se puede prescindir o no de la implementación de un requisito. Puede tomar los valores "Esencial", "Deseable" u "Opcional".
- **Fuente:** Indica la persona o documento del cual ha sido extraído el requisito. Los requisitos pueden proceder de exigencias del cliente o de recomendaciones de expertos en el ámbito que abarca el proyecto.

3.5.1. Requisitos Funcionales

Identificador: RSFU-01	
Título:	Inicio de la aplicación
Descripción:	El usuario inicia la aplicación de forma manual desde su teléfono móvil.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-01

Tabla 3.50: Requisito de software funcional RSFU-01

Identificador: RSFU-02	
Título:	Comprobación de bloqueo de usuario al iniciar la aplicación
Descripción:	La aplicación deberá comprobar si el usuario se encuentra en la lista negra del sistema antes de iniciarse.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-01

Tabla 3.51: Requisito de software funcional RRSFU-02

Identificador: RSFU-03	
Título:	Comprobación tiques usados al iniciar la aplicación
Descripción:	La aplicación deberá actualizar la información de los tiques usados por el usuario (si estuviera desactualizada) antes de iniciarse.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-01

Tabla 3.52: Requisito de software funcional RSFU-03

Identificador: RSFU-04	
Título:	Adquisición de tiques
Descripción:	La aplicación permite al usuario adquirir tiques desde su teléfono móvil.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-02

Tabla 3.53: Requisito de software funcional RSFU-04

Identificador: RSFU-05	
Título:	Listado de tiques sin usar
Descripción:	La aplicación muestra un listado con los tiques adquiridos sin usar que tiene almacenados en su teléfono móvil.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-03

Tabla 3.54: Requisito de software funcional RSFU-05

Identificador: RSFU-06	
Título:	Listado de tiques usados en el día actual
Descripción:	La aplicación permite al usuario consultar los tiques adquiridos usados en el día actual que tiene almacenados en su teléfono móvil.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-04

Tabla 3.55: Requisito de software funcional RSFU-06

Identificador: RSFU-07	
Título:	Establecimiento de código de seguridad
Descripción:	Al iniciar la aplicación por primera vez se pedirá un código de seguridad que se utilizará para proteger la aplicación de usuarios no autorizados.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-05

Tabla 3.56: Requisito de software funcional RSFU-07

Identificador: RSFU-08	
Título:	Uso de tiques para acceder al servicio de transporte público
Descripción:	La aplicación permite al usuario utilizar los tiques adquiridos no usados que tiene almacenados en su teléfono móvil para acceder al servicio de transporte público correspondiente.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-06

Tabla 3.57: Requisito de software funcional RSFU-08

Identificador: RSFU-09	
Título:	Uso de tiques para salir del servicio de transporte público
Descripción:	La aplicación permite al usuario utilizar los tiques adquiridos usados que tiene almacenados en su teléfono móvil para salir del servicio de transporte público.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-07

Tabla 3.58: Requisito de software funcional RSFU-09

Identificador: RSFU-10	
Título:	Envío de tiques al revisor del servicio de transporte público
Descripción:	La aplicación permite al usuario enviar los tiques adquiridos usados que tiene almacenados en su teléfono móvil al revisor del servicio de transporte público si fuera necesario.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-08

Tabla 3.59: Requisito de software funcional RSFU-10

Identificador: RSFU-11	
Título:	Refresco de tiques
Descripción:	La aplicación permite al usuario refrescar los tiques adquiridos no caducados.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-09

Tabla 3.60: Requisito de software funcional RSFU-11

Identificador: RSFU-12	
Título:	Cambio de código de seguridad
Descripción:	La aplicación permite al usuario cambiar el código de seguridad establecido.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-10

Tabla 3.61: Requisito de software funcional RSFU-12

Identificador: RSFU-13	
Título:	Cierre de la aplicación
Descripción:	El usuario puede cerrar la aplicación.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-13

Tabla 3.62: Requisito de software funcional RSFU-13

Identificador: RSFU-14	
Título:	Consulta de todos los tiques adquiridos
Descripción:	La aplicación permite al usuario consultar todos los tiques adquiridos.
Prioridad:	Media
Necesidad:	Esencial
Fuente:	RUC-11

Tabla 3.63: Requisito de software funcional RSFU-14

Identificador: RSFU-15	
Título:	Consulta de todos los tiques adquiridos sin usar
Descripción:	La aplicación permite al usuario consultar todos los tiques adquiridos no usados.
Prioridad:	Media
Necesidad:	Esencial
Fuente:	RUC-11

Tabla 3.64: Requisito de software funcional RSFU-15

Identificador: RSFU-16	
Título:	Consulta de todos los tiques adquiridos usados
Descripción:	La aplicación permite al usuario consultar todos los tiques adquiridos usados.
Prioridad:	Media
Necesidad:	Esencial
Fuente:	RUC-11

Tabla 3.65: Requisito de software funcional RSFU-16

Identificador: RSFU-17	
Título:	Consulta si el usuario está bloqueado
Descripción:	La aplicación permite al usuario consultar si el sistema le ha bloqueado.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-12

Tabla 3.66: Requisito de software funcional RSFU-17

Identificador: RSFU-18	
Título:	Detección de lector NFC
Descripción:	Cuando el usuario vaya a enviar un tique, la aplicación detectará cuando el lector entra dentro del rango de acción para realizar el envío.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-05, RUC-06 y RUC-07

Tabla 3.67: Requisito de software funcional RSFU-18

3.5.2. Requisitos de Interfaz

Identificador: RSIN-01	
Título:	Información sobre el resultado de una operación
Descripción:	La aplicación deberá mostrar al usuario mensajes que indiquen los resultados de las operaciones que se han llevado a cabo.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-15

Tabla 3.68: Requisito de software de interfaz RSIN-01

Identificador: RSIN-02	
Título:	Pantalla principal
Descripción:	La pantalla principal de la aplicación mostrará tres pestañas: «Comprar Ticket», «Tickets» y «Tickets Usados», además de un botón que permite acceder al menú de opciones.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-02, RUC-03 y RUC-04

Tabla 3.69: Requisito de software de interfaz RSIN-02

Identificador: RSIN-03	
Título:	Pantalla de adquisición de tiques
Descripción:	Una vez seleccionada en el menú principal la pestaña «Comprar Tickets» se muestra la pantalla de adquisición de tiques que permite: ▪ Seleccionar el tipo de transporte para el que se desea adquirir el tique (Cercanias, Metro, Bus o Todos). ▪ Seleccionar el tipo de tique que se quiere adquirir (Sencillo o Abono). ▪ Seleccionar la cantidad de tiques que se desea adquirir (de 1 a 9 en tiques de tipo Sencillo o 1 para Abono).
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-02

Tabla 3.70: Requisito de software de interfaz RSIN-03

Identificador: RSIN-04	
Título:	Pantalla de consulta/uso de tiques
Descripción:	Una vez seleccionada en el menú principal la pestaña «Comprar Tickets» se muestra la pantalla de adquisición de tiques que permite seleccionar el tique que el usuario desea utilizar.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-03 y RUC-06

Tabla 3.71: Requisito de software de interfaz RSIN-04

Identificador: RSIN-05	
Título:	Pantalla de consulta/uso de tiques usados
Descripción:	Una vez seleccionada en el menú principal la pestaña «Tickets Usados» se muestra la pantalla de consulta/uso de tiques usados que permite al usuario tanto consultar los tiques que ha usado ese día como seleccionar el tique que desea usar para salir del servicio de transporte público o que quiere enviar al revisor.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-04, RUC-07 y RUC-08

Tabla 3.72: Requisito de software de interfaz RSIN-05

Identificador: RSIN-06	
Título:	Pantalla de menú opciones
Descripción:	Una vez pulsado en el menú principal el botón de opciones se muestra la pantalla del menú de opciones que permite: ▪ Refrescar los tiques adquiridos por el usuario. ▪ Cambiar el código de seguridad de la aplicación. ▪ Mostrar el registro de todos los tiques adquiridos por el usuario. ▪ Comprobar si el usuario está bloqueado por el sistema. ▪ Mostrar información sobre la aplicación.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-09, RUC-10, RUC-11 y RUC-12

Tabla 3.73: Requisito de software de interfaz RSIN-06

Identificador: RSIN-07	
Título:	Pantalla de introducción de código de seguridad
Descripción:	La pantalla se introducción de seguridad se mostrará tanto en el momento de establecer el código de seguridad como en el momento que la aplicación lo requiera para realizar alguna operación.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-05 y RUR-16

Tabla 3.74: Requisito de software de interfaz RSIN-07

Identificador: RSIN-08	
Título:	Idioma de las interfaces
Descripción:	Todas las interfaces de la aplicación están disponibles tanto en inglés como en español. La selección de idioma se realiza en función del idioma establecido en el teléfono.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-02

Tabla 3.75: Requisito de software de interfaz RSIN-08

Identificador: RSIN-09	
Título:	Formato de envío del mensaje NFC para acceder a un servicio de transporte público
Descripción:	Los tiques enviados a través de NFC al lector para acceder a un servicio de transporte público deben respetar el siguiente formato: *«0 +] + identificador del tique +] + tipo de transporte +] + tipo de tique +] + imei +] + fecha de compra del tique +] + fecha de caducidad del tique +] + firma del tique».*
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-06 y RUR-08

Tabla 3.76: Requisito de software de interfaz RSIN-09

Identificador: RSIN-10	
Título:	Formato de envío del mensaje NFC para salir de un servicio de transporte público
Descripción:	Los tiques enviados a través de NFC al lector para salir de un servicio de transporte público deben respetar el siguiente formato: *«1 +] + identificador del tique +] + tipo de transporte +] + tipo de tique +] + imei +] + fecha de compra del tique +] + fecha de caducidad del tique +] + firma del tique +] + hash de la fecha de uso y un código por defecto».*
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-07 y RUR-08

Tabla 3.77: Requisito de software de interfaz RSIN-10

Identificador: RSIN-11	
Título:	Formato de envío del mensaje NFC para enviar tique al revisor de un servicio de transporte público
Descripción:	Los tiques enviados a través de NFC al revisor de un servicio de transporte público deben respetar el siguiente formato: *«2 +] + identificador del tique +] + tipo de transporte +] + tipo de tique +] + imei +] + fecha de compra del tique +] + fecha de caducidad del tique +] + firma del tique +] + hash de la fecha de uso y el código indicado por el revisor».*
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUC-08 y RUR-08

Tabla 3.78: Requisito de software de interfaz RSIN-11

Identificador: RSIN-12	
Título:	Comunicación con el servidor
Descripción:	La comunicación con el servidor se realiza a través de servicios web REST.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-10

Tabla 3.79: Requisito de software de interfaz RSIN-12

3.5.3. Requisitos de Operación

Identificador: RSOP-01	
Título:	Base de datos en el servidor con PostgreSQL
Descripción:	La base de datos del servidor será creada y montada con PostgreSQL 9.1.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-19

Tabla 3.80: Requisito de software de operación RSOP-01

Identificador: RSOP-02	
Título:	Base de datos en el teléfono móvil con SQLite
Descripción:	La base de datos del teléfono móvil será creada y montada con SQLite 3.4.0 o superior.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-13

Tabla 3.81: Requisito de software de operación RSOP-02

Identificador: RSOP-03	
Título:	Identificador de usuario
Descripción:	La aplicación no cuenta con registro de usuarios y la identificación se producirá a través del número IMEI del teléfono móvil.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-01

Tabla 3.82: Requisito de software de operación RSOP-03

Identificador: RSOP-04	
Título:	Tiempos de respuesta con el servidor
Descripción:	El servidor deberá responder a las peticiones de la aplicación en un tiempo máximo de 4 segundos.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-03

Tabla 3.83: Requisito de software de operación RSOP-04

Identificador: RSOP-05	
Título:	Conectividad del teléfono móvil durante el envío de tiques
Descripción:	El teléfono móvil no necesita disponer de conectividad a Internet para poder enviar un tique.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-06

Tabla 3.84: Requisito de software de operación RSOP-05

Identificador: RSOP-06	
Título:	Vibración tras envío de tique
Descripción:	El teléfono móvil vibrará si el envío del tique se produce correctamente.
Prioridad:	Media
Necesidad:	Deseable
Fuente:	RUR-12

Tabla 3.85: Requisito de software de operación RSOP-06

Identificador: RSOP-07	
Título:	Bloqueo de usuarios debido a uso fraudulento de la aplicación
Descripción:	El sistema bloqueará automáticamente a un usuario si detecta que ha intentado hacer un uso fraudulento del mismo.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-17

Tabla 3.86: Requisito de software de operación RSOP-07

Identificador: RSOP-08	
Título:	Bloquear usuarios debido a que se alcanza el número máximo de intentos de introducción de código de seguridad
Descripción:	El sistema bloqueará automáticamente a un usuario si detecta que se alcanza el número máximo de intentos de introducción de código de seguridad.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-17

Tabla 3.87: Requisito de software de operación RSOP-08

Identificador: RSOP-09	
Título:	Conectividad a Internet a través de redes Wi-Fi
Descripción:	Para poder conectar con los servicios web y llevar a cabo determinadas tareas, el dispositivo contará con conexión a Internet a través de redes Wi-Fi.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-05

Tabla 3.88: Requisito de software de operación RSOP-09

Identificador: RSOP-10	
Título:	Conectividad a Internet a través de redes 3G
Descripción:	Para poder conectar con los servicios web y llevar a cabo determinadas tareas, el dispositivo contará con conexión a Internet a través de redes 3G.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-05

Tabla 3.89: Requisito de software de operación RSOP-10

Identificador: RSOP-11	
Título:	Registro de eventos
Descripción:	El sistema registrará automáticamente los eventos que impliquen una comunicación del cliente con el servidor.
Prioridad:	Media
Necesidad:	Esencial
Fuente:	RUR-18

Tabla 3.90: Requisito de software de operación RSOP-11

Identificador: RSOP-12	
Título:	Servidor Apache Tomcat 7.0
Descripción:	El servidor montará Apache Tomcat 7.0 para permitir el despliegue de los servicios web REST que consume la aplicación móvil.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-10

Tabla 3.91: Requisito de software de operación RSOP-12

Identificador: RSOP-13	
Título:	Servicios web REST programados en Java
Descripción:	Los servicios web REST que consume la aplicación móvil se programarán en Java usando la librería Jersey.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-10

Tabla 3.92: Requisito de software de operación RSOP-13

Identificador: RSOP-14	
Título:	Lectura de NFC usando la librería NFCPy
Descripción:	La lectura de los tiques enviados desde el teléfono móvil se realizarán usando la librería NFCPy.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-08

Tabla 3.93: Requisito de software de operación RSOP-14

Identificador: RSOP-15	
Título:	Acceso remoto a servidor
Descripción:	Se podrá acceder de forma remota al servidor a través de SSH.
Prioridad:	Media
Necesidad:	Deseable
Fuente:	RUR-20

Tabla 3.94: Requisito de software de operación RSOP-15

Identificador: RSOP-16	
Título:	Envío de tiques a través de NFC
Descripción:	El envío de tiques se realizará a través de la tecnología NFC.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-08

Tabla 3.95: Requisito de software de operación RSOP-16

3.5.4. Requisitos de Recursos

Identificador: RSRE-01	
Título:	Sistema operativo del teléfono móvil
Descripción:	La aplicación móvil funcionará en teléfonos móviles con sistema operativo Android versión 4.1.2 o superior.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-04

Tabla 3.96: Requisito de software de recursos RSRE-01

Identificador: RSRE-02	
Título:	Sistema operativo del servidor
Descripción:	El sistema operativo del servidor será Debian 6 o Ubuntu 12.04.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-10 y RUR-19

Tabla 3.97: Requisito de software de recursos RSRE-02

Identificador: RSRE-03	
Título:	Sistema operativo del sistema al que se conecta el lector
Descripción:	El sistema operativo del sistema al que se conecta el lector será Debian 6.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-08

Tabla 3.98: Requisito de software de recursos RSRE-03

Identificador: RSRE-04	
Título:	Python en el sistema al que se conecta el lector
Descripción:	El sistema operativo del sistema al que se conecta el lector deberá tener instalada la versión 2.6 de Python.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-08

Tabla 3.99: Requisito de software de recursos RSRE-04

Identificador: RSRE-05	
Título:	Java en el sistema al que se conecta el lector
Descripción:	El sistema operativo del sistema al que se conecta el lector deberá tener instalada la versión 7 de Java.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-08

Tabla 3.100: Requisito de software de recursos RSRE-05

Identificador: RSRE-06	
Título:	Java en el servidor
Descripción:	El sistema operativo del sistema del servidor deberá tener instalada la versión 7 de Java.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-10

Tabla 3.101: Requisito de software de recursos RSRE-06

Identificador: RSRE-07	
Título:	Lector NFC
Descripción:	El lector NFC que va a utilizar el sistema es el SCL3711 de SCM Microsystem.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-09

Tabla 3.102: Requisito de software de recursos RSRE-07

Identificador: RSRE-08	
Título:	Disponibilidad de NFC
Descripción:	El dispositivo móvil deberá disponer de chip NFC para el poder funcionar correctamente.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-07

Tabla 3.103: Requisito de software de recursos RSRE-08

3.5.5. Requisitos de Comprobación

Identificador: RSCO-01	
Título:	Comprobación de conectividad
Descripción:	Se comprobará si el dispositivo tiene conectividad a Internet ya se a través de redes Wi-Fi como 3G antes de consumir un servicio web.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-05

Tabla 3.104: Requisito de software de comprobación RSCO-01

Identificador: RSCO-02	
Título:	Comprobación de código de seguridad
Descripción:	Se comprobará si el código introducido por el usuario es correcto cada vez que se solicite.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-16

Tabla 3.105: Requisito de software de comprobación RSCO-02

3.5.6. Requisitos de Seguridad

Identificador: RSSE-01	
Título:	Almacenamiento seguro de tiques en el dispositivo móvil
Descripción:	Los tiques almacenados en el dispositivo se almacenarán de forma que la firma quede cifrada y solo pueda utilizarlos el usuario que conoce el código de seguridad.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-14

Tabla 3.106: Requisito de software de seguridad RSSE-01

Identificador: RSSE-02	
Título:	Conexión segura con el servidor
Descripción:	Todas las comunicaciones que se produzcan entre el teléfono móvil y el servidor se llevaran a cabo utilizando el protocolo SSL.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-11

Tabla 3.107: Requisito de software de seguridad RSSE-02

Identificador: RSSE-03	
Título:	Petición de código de usuario
Descripción:	La aplicación móvil deberá solicitar el código de seguridad antes de realizar determinadas operaciones como adquirir tiques, usar tiques, refrescar los tiques, cambiar el código de seguridad y comprobar si el usuario está en la lista negra del sistema.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-16

Tabla 3.108: Requisito de software de seguridad RSSE-03

Identificador: RSSE-04	
Título:	Puertos abiertos en el servidor
Descripción:	El servidor solo tendrá abiertos los puertos 8443 (servicios web con SSL) y 2022 para conexión SSH para prevenir ataques.
Prioridad:	Alta
Necesidad:	Esencial
Fuente:	RUR-10

Tabla 3.109: Requisito de software de seguridad RSSE-04

3.6. Matrices de Trazabilidad

3.6.1. Matriz de Trazabilidad de Requisitos de Usuario a Casos de Uso

	RUC-01	RUC-02	RUC-03	RUC-04	RUC-05	RUC-06	RUC-07	RUC-08	RUC-09	RUC-10	RUC-11	RUC-12	RUC-13
CU-01	X												
CU-02					X								
CU-03		X											
CU-04						X							
CU-05			X										
CU-06				X									
CU-07									X				
CU-08										X			
CU-09											X		
CU-10							X						
CU-11								X					
CU-12												X	
CU-13													X

Tabla 3.110: Matriz de trazabilidad RUC-CU

3.6.2. Matriz de Trazabilidad de Requisitos de Usuario a Requisitos Software

	RUC-01	RUC-02	RUC-03	RUC-04	RUC-05	RUC-06	RUC-07	RUC-08	RUC-09	RUC-10	RUC-11	RUC-12	RUC-13	RUR-01	RUR-02	RUR-03	RUR-04	RUR-05	RUR-06	RUR-07	RUR-08	RUR-09	RUR-10	RUR-11	RUR-12	RUR-13	RUR-14	RUR-15	RUR-16	RUR-17	RUR-18	RUR-19	RUR-20
RSFU-01	X																																
RSFU-02	X																																
RSFU-03	X																																
RSFU-04		X																															
RSFU-05			X																														
RSFU-06				X																													
RSFU-07					X																												
RSFU-08						X																											
RSFU-09							X																										
RSFU-10								X																									
RSFU-11									X																								
RSFU-12										X																							
RSFU-13													X																				
RSFU-14											X																						
RSFU-15											X																						
RSFU-16											X																						
RSFU-17												X																					
RSFU-18					X	X	X																										
RSIN-01																												X					
RSIN-02		X	X	X																													
RSIN-03		X																															

	RUC-01	RUC-02	RUC-03	RUC-04	RUC-05	RUC-06	RUC-07	RUC-08	RUC-09	RUC-10	RUC-11	RUC-12	RUC-13	RUR-01	RUR-02	RUR-03	RUR-04	RUR-05	RUR-06	RUR-07	RUR-08	RUR-09	RUR-10	RUR-11	RUR-12	RUR-13	RUR-14	RUR-15	RUR-16	RUR-17	RUR-18	RUR-19	RUR-20
RSIN-04			X			X																										
RSIN-05				X			X	X																								
RSIN-06									X	X	X	X																				
RSIN-07					X																							X			
RSIN-08															X																	
RSIN-09						X															X											
RSIN-10							X														X											
RSIN-11								X													X											
RSIN-12																							X									
RSOP-01																															X	
RSOP-02																										X						
RSOP-02														X																		
RSOP-04																X																
RSOP-05																			X													
RSOP-06																									X							
RSOP-07																														X		
RSOP-08																														X		
RSOP-09																		X														
RSOP-10																		X														
RSOP-11																															X	
RSOP-12																							X									
RSOP-13																							X									
RSOP-14																					X											

	RUC-01	RUC-02	RUC-03	RUC-04	RUC-05	RUC-06	RUC-07	RUC-08	RUC-09	RUC-10	RUC-11	RUC-12	RUC-13	RUR-01	RUR-02	RUR-03	RUR-04	RUR-05	RUR-06	RUR-07	RUR-08	RUR-09	RUR-10	RUR-11	RUR-12	RUR-13	RUR-14	RUR-15	RUR-16	RUR-17	RUR-18	RUR-19	RUR-20
RSOP-15																																	X
RSOP-16																					X												
RSRE-01																	X																
RSRE-02																							X									X	
RSRE-03																					X												
RSRE-04																					X												
RSRE-05																					X												
RSRE-06																							X										
RSRE-07																						X											
RSRE-08																				X													
RSCO-01																		X															
RSCO-02																													X				
RSSE-01																											X						
RSSE-02																								X									
RSSE-03																													X				
RSSE-04																							X										

Tabla 3.111: Matriz de trazabilidad RU-RS

Parte III

Diseño e Implementación del prototipo

4

Diseño del sistema

Tomando como base el análisis realizado en el capítulo 3, se llevará a cabo la fase de diseño del sistema donde se describirá en detalle cómo va a ser implementado y se justificarán cada una de las decisiones tomadas.

Con esta finalidad, este capítulo recoge distintas secciones que incluirán diagramas y explicaciones tanto del funcionamiento del sistema en conjunto como del diseño de cada uno de los subsistemas que lo componen.

4.1. Arquitectura del sistema

El sistema está compuesto de tres partes que funcionan en conjunto para proporcionar la funcionalidad completa al mismo: el servidor, la aplicación móvil y la aplicación de lectura NFC.

- El servidor aloja una base de datos y ofrece a las aplicaciones cliente los servicios web necesarios para que el sistema funcione correctamente.
- La aplicación móvil es la herramienta con la que el usuario interactua con el sistema para utilizar los servicios ofrecidos por el mismo.
- La aplicación de lectura de NFC lee el tique enviado por el usuario al lector, valida dicho tique y actualiza la información correspondiente a ese tique en la base de datos del servidor a través de un servicio web.

El sistema ha sido diseñado en base a una arquitectura Cliente-Servidor de manera que el servidor provea a las aplicaciones clientes de servicios web que permitirán que estas aplicaciones puedan acceder y/o modificar la información sin interactuar directamente con la base datos. Es decir, los servicios web harán de intermediario entre las aplicaciones y la base de datos.

La arquitectura Cliente-Servidor puede considerarse como una arquitectura distribuida que permite a los usuarios finales obtener acceso a la información en forma transparente incluso en entornos multiplataforma. En este modelo, las tareas se reparten entre los servidores y los clientes. Un cliente realiza peticiones al servidor que se que encarga de enviarle la respuesta. La capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes

las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema.

- **Cliente:** Los clientes se encargan de gestionar la comunicación con el servidor, de solicitar un servicio concreto y de recibir los datos enviados por éste. Además son la herramienta que presenta al usuario los datos en pantalla y que le ofrece los comandos necesarios para utilizar las prestaciones que ofrece el servidor.

- **Servidor:** El servidor es el proceso encargado de atender a los múltiples clientes que hacen peticiones de algún recurso administrado por él.

En la figura 4.1 se muestra la arquitectura Cliente-Servidor del sistema. La parte del cliente se corresponde con las aplicaciones que consumirán los servicios web ofrecidos por la parte del servidor.

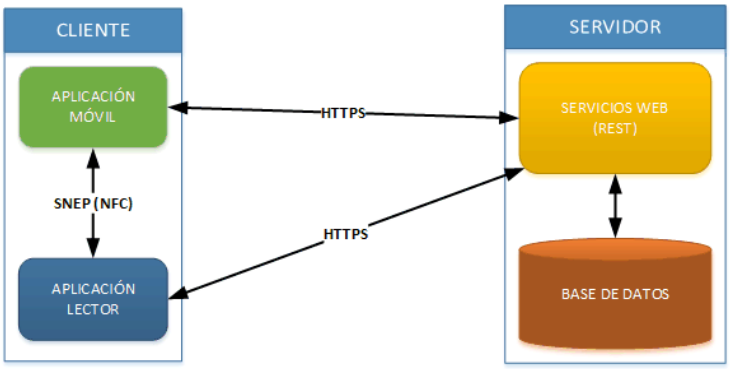

Figura 4.1: Arquitectura Cliente-Servidor del sistema

Los servicios web del servidor han sido diseñados utilizando la arquitectura REST que utiliza el protocolo HTTP para comunicarse. Esta decisión de diseño será justificada más adelante en la sección correspondiente.

Por otro lado, Android está diseñado de forma que las aplicaciones desarrolladas sigan el patrón de arquitectura Modelo-Vista-Controlador (MVC) cuya principal ventaja consiste en separar los datos de una aplicación, la interfaz de usuario y la lógica de negocio en tres componentes distintos que se relacionarán para al final tener como resultado la aplicación. Para ello, este modelo propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador:

- **Modelo:** Es la representación específica de la información con la cual operará la aplicación. Envía a la vista aquella parte de la información que en cada momento se le solicita para que sea mostrada. Las peticiones de acceso o manipulación de información llegan al modelo a través del controlador. En Android, el modelo puede obtenerse tanto de la base de datos local del sistema (SQLite) como a través de servicios web.

- **Vista:** Presenta el modelo en un formato adecuado para interactuar con el usuario (interfaces de usuario). En Android, las interfaces de usuario se construyen usando XML.

- **Controlador:** Responde a eventos (usualmente acciones del usuario) e invoca peticiones al modelo cuando se hace alguna solicitud sobre la información. Se podría decir que el controlador hace de intermediario entre la vista y el modelo. En Android, el controlador lo forman todas las clases Java que permiten utilizar las interfaces y permiten desplegar y consumir información de/para el usuario.

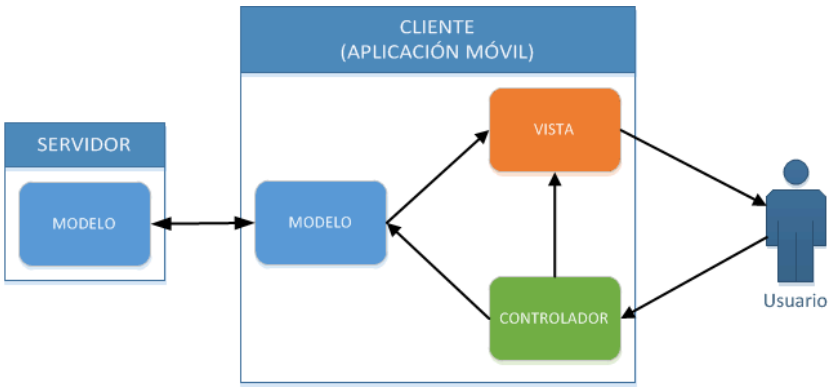

Figura 4.2: Arquitectura MVC del sistema

La aplicación móvil mostrará las interfaces con las que interactuará el usuario, que se correspondería con la parte de la vista. La aplicación móvil también implementará la parte del controlador y se encargará de obtener la información del modelo ya sea directamente desde la propia base de datos de Android o a través de servicios web.

4.2. Diagrama de despliegue

El diagrama de despliegue es un tipo de diagrama UML que permite representar la disposición de los elementos hardware necesarios para ejecutar el sistema, así como los diversos entornos de ejecución y artefactos software requeridos para este fin. El diagrama de despliegue del sistema es el siguiente:

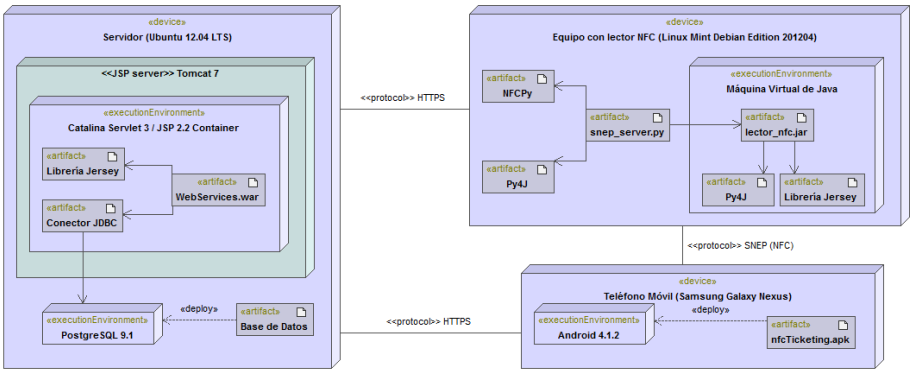

Figura 4.3: Diagrama de despliegue del sistema

Se dispone de un equipo que tiene instalado el sistema operativo **Ubuntu 12.04 LTS** que hará la función de servidor. Además se ha instalado el sistema gestor de base de datos **PostgreSQL 9.1** con el que se crea y gestiona la base de datos (**Base de Datos**), el servidor web **Apache Tomcat 7.0.26** que es un contenedor de servlets y JSPs que ejecutará los servicios web que emplearán las aplicaciones cliente y la **Máquina Virtual de Java (JVM) 1.7.0_15** necesaria para el correcto funcionamiento de Tomcat. A su vez, los servicios web necesitarán la **librería Jersey** que permite la implementación de servicios web RESTful en Java y el **conector JDBC** para acceder a bases de datos PostgreSQL.

Por otro lado, se tiene un equipo que tiene instalado la distribución **Linux Mint Debian Edition 201204** que está basada en Debian 6. Este equipo tendrá conectado el lector NFC y necesitará tener instalado **Python 2.6**, la librería **NFCPy** que permite leer los datos recibidos por el lector y la librería **Py4J** que permite a Python acceder de forma dinámica a objetos que se ejecutan en la Máquina Virtual de Java y la propia Máquina Virtual de Java (JVM) 1.7.0_03. En la Máquina Virtual de Java se ejecutará la aplicación que permite validar los tiques (**lector_nfc.jar**) así como comunicarse a través de los servicios web con el servidor para actualizar la información de los tiques. Esta aplicación necesitará la la librería **Jersey** para consumir los servicios web ofrecidos por el servidor y la librería **Py4J** para permitir que Python pueda acceder a los métodos de la aplicación Java. La aplicación **snep_server.py** permite leer el tique enviado por el usuario desde su teléfono móvil y validar el mismo a través de la aplicación en Java.

Por último, se cuenta con un teléfono móvil Samsung Galaxy Nexus con la versión de **Android 4.1.2** que tiene instalada la aplicación móvil **nfcTicketing.apk**.

4.3. Funcionamiento del sistema

El funcionamiento del sistema en conjunto es prácticamente igual al funcionamiento de los tiques físicos de transporte público de hoy en día. El sistema se diseño en base a este paradigma de manera que la aplicación móvil únicamente facilite y acelere este proceso sin cambiarlo notablemente.

En la figura 4.4 que se muestra a continuación se detallan todos los pasos que se siguen desde que el usuario adquiere un tique con la aplicación hasta que lo utiliza para poder acceder a un servicio de transporte público:

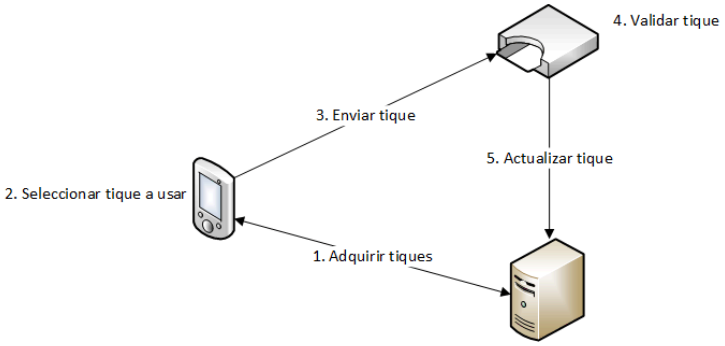

Figura 4.4: Diagrama de funcionamiento del sistema para acceder al servicio

1. El usuario adquiere los tiques desde la pestaña «Comprar Tickets» de la aplicación móvil. El servidor recibirá la petición, generará los tiques pedidos por el usuario firmando cada uno de ellos, los almacenará en la base de datos y los enviará al teléfono del usuario. El teléfono los almacenará cifrando la firma de manera que únicamente puedan ser utilizados por la aplicación. Los mecanismos de seguridad empleados para proteger el sistema de un uso indebido serán explicados más adelante.

2. El usuario selecciona el tique que desea usar en la aplicación móvil.

3. Una vez seleccionado el tique, el usuario aproxima su teléfono móvil al lector NFC para enviarlo.

4. La aplicación del lector recibe y valida el tique. La validación del tique se realiza "offline", esto es que el lector no se tiene que comunicar con el servidor para verificar si un tique es correcto. Si el tique es correcto permite el acceso del usuario. En caso contrario no permite el acceso del usuario al servicio de transporte público. Si un usuario envía un tique no válido (firma del tique incorrecta) se considera que el usuario ha intentando falsificar un tique. Si un usuario envía 3 tiques no válidos a un lector será incluido en la lista negra del sistema y no podrá utilizar la aplicación móvil.

5. La aplicación del lector se comunica con el servidor a través de servicios web para actualizar la información del tique enviado indicando la fecha de uso. Si el tique no fuera válido, se actualizarán el número de intentos de usar un tique falso del usuario.

En la figura 4.5 se explican todos los pasos que se siguen para que un usuario pueda salir de un servicio de transporte público usando la aplicación:

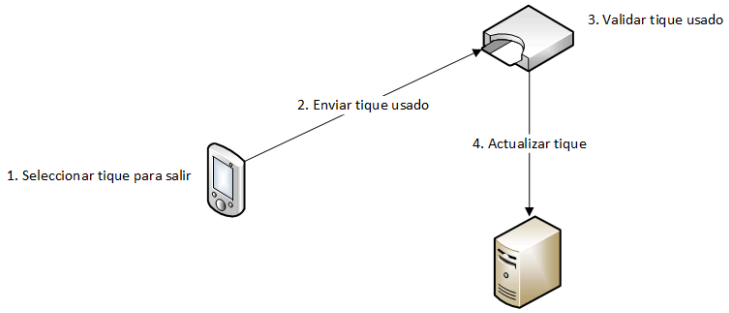

Figura 4.5: Diagrama de funcionamiento del sistema para salir del servicio

1. El usuario selecciona el tique que desea usar para salir en la aplicación móvil. Se considera que el usuario utilizará para salir el tique que empleo para acceder al servicio.

2. Una vez seleccionado el tique, el usuario aproxima su teléfono móvil al lector NFC para enviarlo.

3. La aplicación del lector recibe y valida el tique para salir. La validación del tique se realiza "offline", esto es que el lector no se tiene que comunicar con el servidor para verificar si un tique es correcto. Si el tique es correcto permite que el usuario salga del servicio de transporte. En caso contrario no permite que el usuario abandone el servicio de transporte público. Si un usuario envía un tique no válido (firma del tique incorrecta) se considera que el usuario ha intentando falsificar un tique. Si un usuario envía 3 tiques no válidos a un lector será incluido en la lista negra del sistema y no podrá utilizar la aplicación móvil.

4. La aplicación del lector se comunica con el servidor a través de servicios web para actualizar la información del tique enviado indicando que el tique se ha empleado para salir. Si el tique no fuera válido, se actualizarán el número de intentos de usar un tique falso del usuario.

5

Diseño de la aplicación móvil

En el siguiente apartado se describirá el diseño y la implementación de la aplicación móvil desarrollada. Se incluirán diversos diagramas de clases de facilitarán al lector la comprensión del diseño de la aplicación.

5.1. Diagrama de clases

Un diagrama de clases describe la estructura de una aplicación mostrando las clases con sus métodos, atributos y visibilidad de los mismos (*public*, *private* o *protected*), y las relaciones entre ellas. En este caso, no se van a mostrar los atributos y métodos en el diagrama general por falta de espacio y hacer más sencilla la visión global de la aplicación, pero serán comentadas en el siguiente apartado.

En el diagrama, podrán encontrarse tres tipos de relaciones distintas:

- **Asociaciones:** Flechas trazadas con líneas continuas y la punta abierta, que indican que dentro de una clase existe un atributo del tipo de la clase a la que se está apuntando. El nombre del atributo se muestra junto a la relación, y el valor de visibilidad de ese atributo se coloca delante del nombre en forma de signo («+» para atributos públicos, «-» para privados, «#» para atributos protegidos o « » para atributos de paquete).
- **Dependencias:** Flechas trazadas con líneas discontinuas, la punta abierta y sin ningún texto junto a ellas. Indican que una clase depende de la creación y uso de objetos de otra clase, aunque no los almacene como atributos.
- **Asociaciones de propiedad de elementos:** Líneas continuas y con un signo más dentro de una circunferencia en el lado del propietario, que indican que una clase está anidada a otra. La clase anidada solo puede ser utilizada por la clase a la que está anidada.

Para mayor comodidad y facilidad de lectura se ha desglosado el diagrama de clases por partes. Las clases serán definidas con más detalle en la siguiente

sección. Este primer diagrama representa la relación de los paquetes que contienen las clases de aplicación entre sí. Las clases se han estructura en paquetes que contienen las clases relacionadas con una determinada funcionalidad.

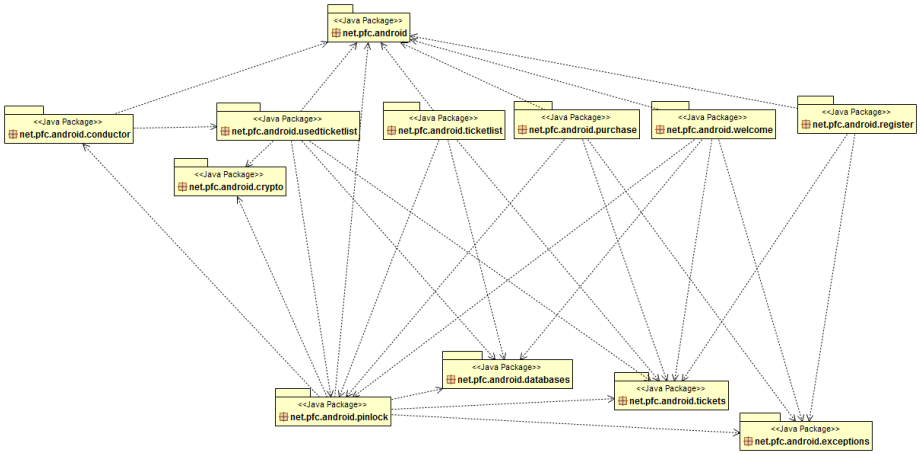

Figura 5.1: Diagrama de clases de los paquetes

- **net.pfc.android:** Contiene la clase principal de la aplicación.
- **net.pfc.android.conductor:** Contiene la clase relacionada con la generación de un tique para mostrar al revisor.
- **net.pfc.android.crypto:** Contiene la clase que permite realizar todas las operaciones criptográficas en la aplicación.
- **net.pfc.android.databases:** Contiene la clase que permite gestionar la base de datos en la aplicación.
- **net.pfc.android.exceptions:** Contiene la clase que permite identificar errores en la aplicación.
- **net.pfc.android.pinlock:** Contiene la clase que permite gestionar el establecimiento y la introducción del código de seguridad de la aplicación.
- **net.pfc.android.purchase:** Contiene la clase que permite adquirir tiques desde la aplicación.
- **net.pfc.android.register:** Contiene las clases que permiten mostrar el registro de todos los tiques adquiridos por el usuario ya sean usados o sin usar en la aplicación.
- **net.pfc.android.ticketlist:** Contiene las clases que permiten mostrar al usuario el listado con los tiques adquiridos sin usar y seleccionarlos para usarlos.
- **net.pfc.android.tickets:** Contiene las clases que permiten realizar una representación de los objetos tique para facilitar la gestión de los mismos por la aplicación.

- **net.pfc.android.usedticketlist:** Contiene las clases que permiten mostrar al usuario el listado con los tiques adquiridos usados en el día actual y seleccionarlos para usarlos ya sea para salir o para enviar al revisor.

- **net.pfc.android.welcome:** Contiene las clases que permiten mostrar al usuario la pantalla de inicio de la aplicación y realizar la operaciones necesarias durante el mismo.

A continuación se muestran los distintos diagramas de clases que muestran las clases que intervienen en un determinada funcionalidad de la aplicación.

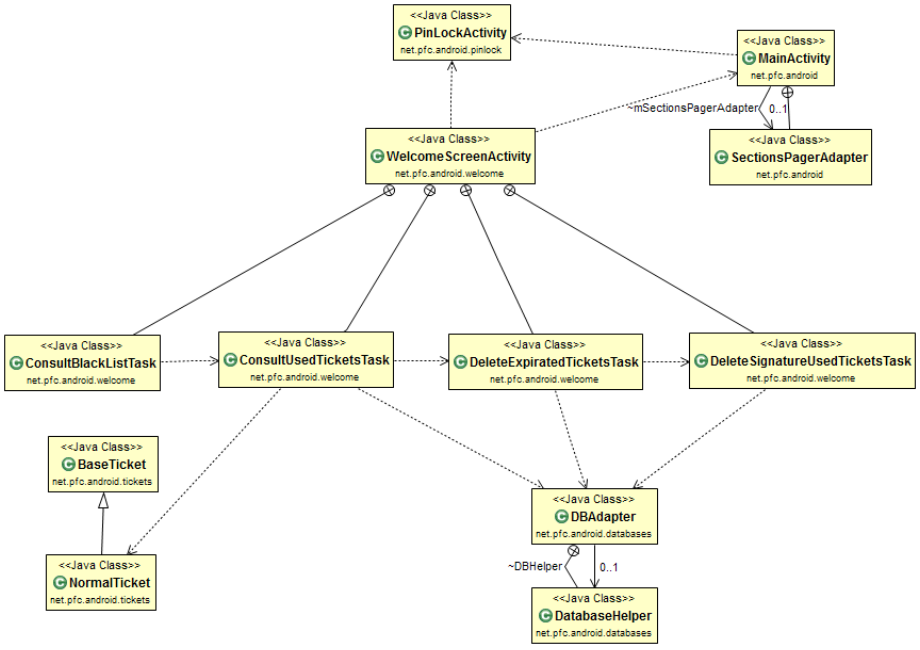

Figura 5.2: Diagrama de clases de la pantalla de bienvenida

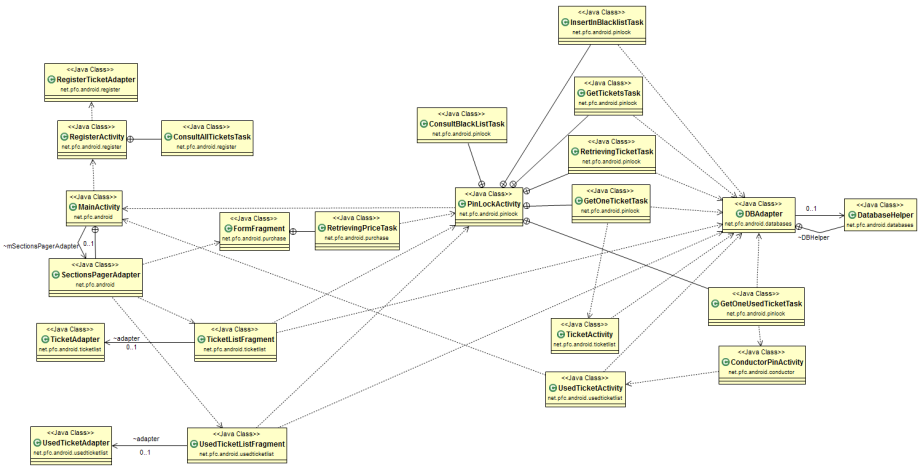

Figura 5.3: Diagrama de clases de la pantalla de principal

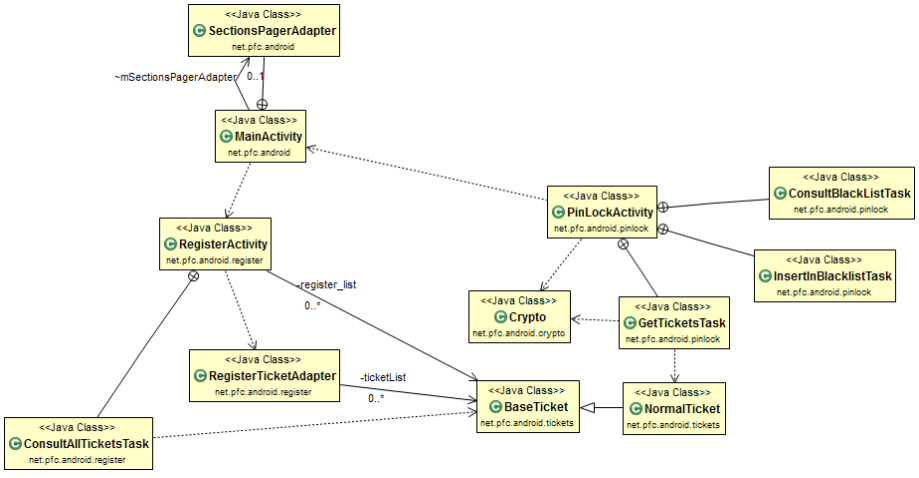

Figura 5.4: Diagrama de clases de la pantalla del menú de opciones

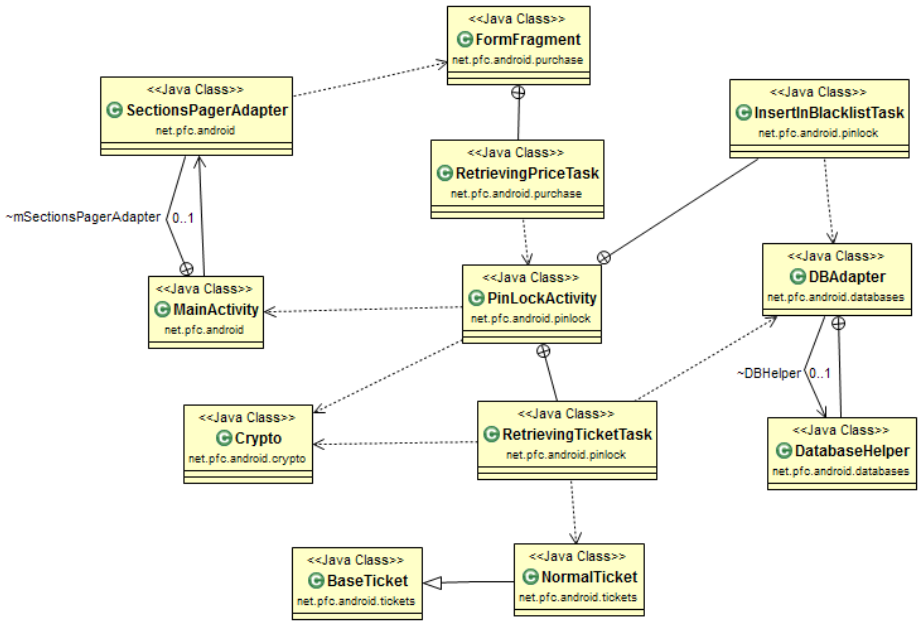

Figura 5.5: Diagrama de clases de la pantalla de «Comprar Ticket»

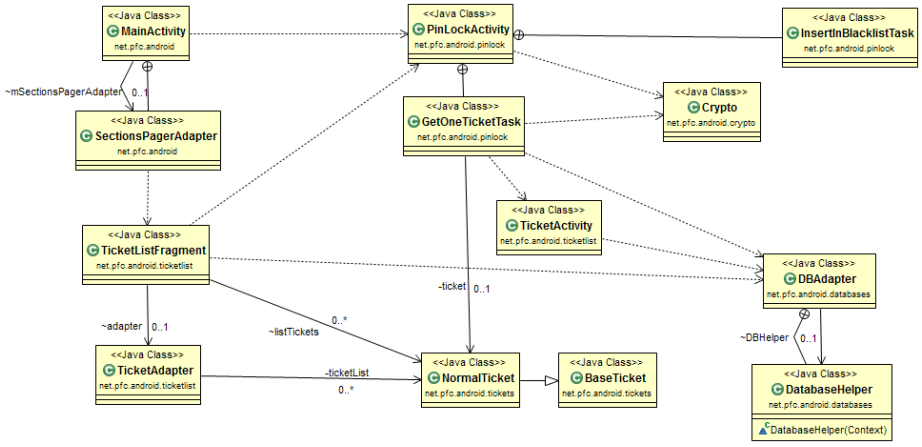

Figura 5.6: Diagrama de clases de la pantalla de «Mis Tickets»

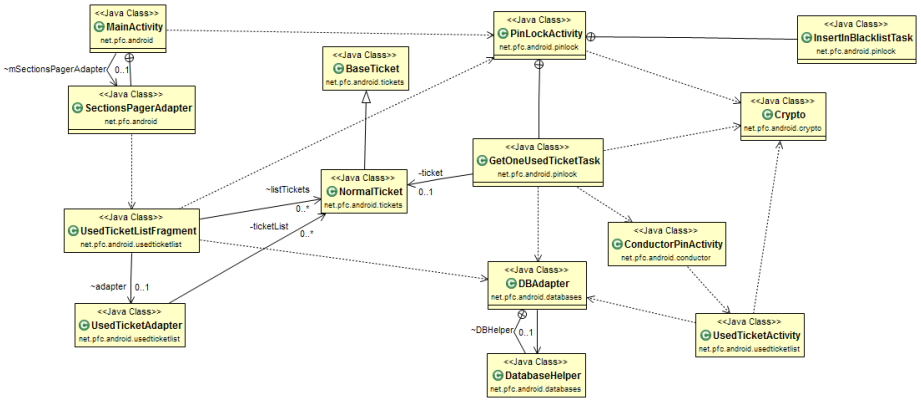

Figura 5.7: Diagrama de clases de la pantalla de «Tickets Usados»

5.2. Definición de las clases

La descripción completa de todas las clases, atributos y métodos puede encontrarse detallada en la documentación *Javadoc* del proyecto. No obstante, en el siguiente apartado se va a explicar la función de cada clase que aparece en el diagrama comentando los atributos y métodos más relevantes.

Clase `MainActivity`

Clase principal de la aplicación que muestra la pantalla principal y permite al usuario navegar entre las tres funcionalidades principales que ofrece el sistema: adquirir tiques, mostrar tiques adquiridos sin usar y mostrar tiques usados en el día actual. Las funcionalidades se encuentran separadas en tres pestañas diferentes y esta clase permite que se muestre la opción correcta en función de la selección del usuario. Dentro de la propia clase se crea la clase `SectionsPagerAdapter` que permitirá personalizar la navegación de las pestañas. Esta clase también permite al usuario acceder al menú de opciones.

Contiene el método `checkConnectivity` que permite realizar una comprobación de la conectividad a Internet del teléfono móvil antes de intentar cualquier operación que requiera de ella.

Clase `ConductorPinActivity`

Clase que muestra al usuario la interfaz que le permite introducir el código con el que se generará el tique que o le permitirá salir del servicio de transporte público o enviará al revisor si se le solicitara. Simplemente recoge el código introducido por el usuario y lo envía junto con la información necesaria a la clase que generará el tique.

Clase `Crypto`

Clase que recoge toda la funcionalidad criptográfica de la aplicación. Está clase permite derivar claves a partir del código introducido por el usuario, generar claves AES, cifrar y descifrar tiques, generar resúmenes...

Los atributos más destacables de la clase son:

- `KEY_LENGTH`: Indica el tamaño de la clave generada en bits.
- `ITERATION_COUNT`: Indica el número de iteraciones que se realizan en el proceso de derivación de la clave.
- `SALT_LENGTH`: Indica el tamaño del *salt* generado en bytes.
- `DELIMITER`: Indica el carácter que servirá como separador cuando se cifre un texto ya que el resultado del cifrado será `salt+DELIMITER+texto_cifrado`.

Los métodos más importantes de la clase son:

- `deriveKeyPkcs12`: Permite derivar una clave AES de longitud `KEY_LENGTH` en `ITERATION_COUNT` iteraciones utilizando el algoritmo especificado en PKCS#12.
- `encryptPkcs12`: Cifra utilizando el algoritmo `AES/CBC/PKCS5Padding` una cadena de texto con la clave derivada a partir de una contraseña dada usando el algoritmo especificado en PKCS#12.
- `decryptPkcs12`: Descifra utilizando el algoritmo `AES/CBC/PKCS5Padding` una cadena de texto con la clave derivada a partir de una contraseña dada usando el algoritmo especificado en PKCS#12.
- `encryptAes`: Cifra utilizando el algoritmo `AES/CBC/PKCS5Padding` una cadena de texto con la clave dada.
- `decryptAes`: Descifra utilizando el algoritmo `AES/CBC/PKCS5Padding` una cadena de texto con la clave dada.
- `getHash`: Obtiene el resumen de una cadena de texto usando el algoritmo indicado (`SHA-512` en este caso).
- `generateK`: Genera una clave AES aleatoria de longitud `KEY_LENGTH`.

Clase `DBAdapter`

Clase que implementa toda la funcionalidad relacionada con la inserción, modificación, borrado y consulta de los tiques almacenados en la base de datos de la aplicación. Dentro de la propia clase se crea la clase `DatabaseHelper` que permitirá crear la base de datos y la tabla que necesita la aplicación.

Los métodos más destacados de la clase son:

- `open`: Permite abrir la base de datos de la aplicación.
- `close`: Permite cerrar la base de datos de la aplicación.
- `insertTicket`: Inserta un tique dado en la tabla de la base de datos.
- `deleteTicket`: Borra el tique indicado en la tabla de la base de datos.
- `deleteAllTicket`: Borra todos los tiques de la tabla de la base de datos.

- `getAllTickets`: Obtiene todos los tiques de la tabla de la base de datos.
- `getTodayUsedTickets`: Obtiene todos los tiques usados en el día actual de la base de datos.
- `deleteSignatureUsedTickets`: Borra la firma de todos los tiques usados de la base de datos.
- `getNoUsedTickets`: Obtiene todos los tiques no usados de la base de datos.
- `getTicket`: Obtiene el tique indicado de la base de datos.
- `deleteSignatureUsedTickets`: Borra la firma de el tique indicado de la base de datos.

Clase `ExceptionTypes`

Clase de tipo `enum` que sirve para identificar errores relacionados con la conexión con los servicios web del servidor y que permiten realizar una gestión de los mismos.

Clase `PinLock`

Clase que permite realizar todas las operaciones relacionadas con la introducción del código de seguridad. Esta clase puede ser llamada desde distintos contextos por lo que realiza un control para saber desde donde ha sido llamada y realizar las tareas correspondientes.

Los atributos más destacables de la clase son:

- `URL`: URL que contiene la IP del servidor y el puerto en el que están disponibles los servicios web.
- `DEFAULT_ATTEMPTS`: Número de intentos máximos permitidos de introducción del código de seguridad antes de bloquear la aplicación.
- `TRUSTSTORE_PASSWORD`: Contraseña del almacén de claves que contiene el certificado de clave pública que permite la comunicación con los servicios web del servidor.

Los métodos más destacados de la clase son:

- `managePIN`: En función de ciertos parámetros permite saber desde que contexto ha sido llamada la clase y ejecuta el método correspondiente.
- `setPin`: Método que permite establecer un código de seguridad.
- `buyTicket`: Método que llama a los métodos necesarios para adquirir un tique.
- `selectTicket`: Método que llama a los métodos necesarios para mostrar un tique seleccionado.
- `changePin`: Método que permite cambiar el código de seguridad.

- `refreshTicket`: Método que llama a los métodos necesarios para refrescar los tiques adquiridos por el usuario.
- `consultBlacklist`: Método que llama a los métodos necesarios para comprobar si el usuario está bloqueado.

Dentro de la propia clase se crean varias clases que extienden de `AsyncTask`. Esto permite crear tareas asíncronas que realizan una tarea en segundo plano evitando que la aplicación se bloquee. Estas clases son las que permitirán realizar las llamadas a los servicios web del servidor y son las siguientes:

- `RetrievingTicketTask`: Permite llamar al servicio web que genera los tiques pedidos por el usuario.
- `GetOneTicketTask`: Permite descifrar un tique y prepararlo para su envío.
- `GetOneUsedTicketTask`: Permite descifrar un tique usado y prepararlo para su envío.
- `GetTicketsTask`: Permite llamar al servicio web que envía todos los tiques adquiridos por el usuario.
- `ConsultBlackListTask`: Permite llamar al servicio web que consulta si el usuario está bloqueado.
- `InsertInBlacklistTask`: Permite llamar al servicio web que inserta al usuario en la lista negra.

Clase `FormFragment`

Esta clase extiende de la clase `Fragment`. Los fragmentos en Android son como una una sección modular de una actividad, que tiene su propio ciclo de vida, recibe sus propios eventos de entrada, y que se puede añadir o quitar mientras la actividad se está ejecutando. Este fragmento se mostrará en la pantalla principal cuando el usuario seleccione la pestaña «Comprar Tickets» .

Los atributos más destacables de la clase son:

- `URL`: URL que contiene la IP del servidor y el puerto en el que están disponibles los servicios web.
- `TRUSTSTORE_PASSWORD`: Contraseña del almacén de claves que contiene el certificado de clave pública que permite la comunicación con los servicios web del servidor.

Los métodos más importantes de la clase son:

- `checkConnectivity`: Permite realizar una comprobación de la conectividad a Internet del teléfono móvil antes de intentar cualquier operación que requiera de ella.
- `inputStreamToString`: Permite leer la respuesta recibida como respuesta a la llamada realizada al servicio web.

Dentro de la propia clase se crea una clase que extiende de `AsyncTask`. La tarea asíncrona es la siguiente:

- `RetrievingPriceTask`: Permite llamar al servicio web que obtiene el precio unitario del tipo de tique seleccionado por el usuario.

Clase `RegisterActivity`

Clase que permite mostrar todos los tiques adquiridos por el usuario. El usuario podrá elegir entre verlos todos, ver tiques no usados o ver los tiques usados.

Los atributos más destacables de la clase son:

- `URL`: URL que contiene la IP del servidor y el puerto en el que están disponibles los servicios web.
- `TRUSTSTORE_PASSWORD`: Contraseña del almacén de claves que contiene el certificado de clave pública que permite la comunicación con los servicios web del servidor.

El método más importante de la clase es:

- `checkConnectivity`: Permite realizar una comprobación de la conectividad a Internet del teléfono móvil antes de intentar cualquier operación que requiera de ella.

Dentro de la propia clase se crea una clase que extiende de `AsyncTask`. La tarea asíncrona es la siguiente:

- `ConsultAllTicketsTask`: Permite llamar al servicio web que obtiene todos los tiques adquiridos por el usuario.

Clase `RegisterTicketAdapter`

Clase que permite personalizar los elementos de la lista que se va a mostrar en la clase `RegisterActivity`. La clase extiende de la clase `BaseAdapter` que es un adaptador básico para crear listas.

El atributo más destacable de la clase es:

- `ticketList`: Lista que contiene los elementos que se van a mostrar.

Los métodos más importantes de la clase son:

- `getCount`: Obtiene el número de elementos de la lista.
- `getItem`: Devuelve el elemento de la lista en la posición indicada.
- `getView`: Permite personalizar la vista de cada elemento de la lista.

Clase `TicketActivity`

Clase que muestra un tique sin usar seleccionado por el usuario y lo prepara para su envío a través de NFC.

Los métodos más importantes de la clase son:

- `setTicket`: Método que establece los campos que se mostrarán del tique seleccionado.
- `createNdefMessage`: Crea el mensaje NDEF con los datos del tique que será enviado.
- `onNdefPushComplete`: Crea *handler* que permite recibir el *callback* de la operación de envío y así saber si se ha realizado con éxito.

Clase `TicketAdapter`

Clase que permite personalizar los elementos de la lista de tiques sin usar que se va a mostrar en la clase `TicketListFragment`. La clase extiende de la clase `BaseAdapter` que es un adaptador básico para crear listas.

El atributo más destacable de la clase es:

- `ticketList`: Lista que contiene los elementos que se van a mostrar.

Los métodos más importantes de la clase son:

- `getCount`: Obtiene el número de elementos de la lista.
- `getItem`: Devuelve el elemento de la lista en la posición indicada.
- `getView`: Permite personalizar la vista de cada elemento de la lista.
- `updateTickets`: Permite notificar que la lista ha sufrido cambios y refrescar la vista de la misma.

Clase `TicketListFragment`

Clase que permite mostrar al usuario un listado con los tiques adquiridos sin usar. Este fragmento se mostrará en la pantalla principal cuando el usuario seleccione la pestaña «Mis Tickets».

El atributo más destacable de la clase es:

- `ticketList`: Lista que contiene los elementos que se van a mostrar.

El método más importante de la clase es:

- `onListItemClick`: Obtiene los datos del tique seleccionado por el usuario.

Clase `BaseTicket`

Clase que permite representar un objeto tique básico que solo tiene como atributos el identificador, el tipo de transporte, el tipo de tique, la fecha de compra y la fecha de uso.

Clase `NormalTicket`

Clase que extiende de la clase `BaseTicket` y añade los atributos de fecha de caducidad y firma al tique básico.

Clase `Transport`

Clase de tipo `enum` que indica los tipos de transporte para los que se pueden adquirir tiques desde la aplicación.

Clase `Type`

Clase de tipo `enum` que indica los tipos de tique que se pueden adquirir desde la aplicación.

Clase `UsedTicketActivity`

Clase que muestra un tique usado seleccionado por el usuario y lo prepara para su envío a través de NFC.

Los métodos más importantes de la clase son:

- `setTicket`: Método que establece los campos que se mostrarán del tique seleccionado.
- `createNdefMessage`: Crea el mensaje NDEF con los datos del tique que será enviado.
- `onNdefPushComplete`: Crea *handler* que permite recibir el *callback* de la operación de envío y así saber si se ha realizado con éxito.

Clase `UsedTicketAdapter`

Clase que permite personalizar los elementos de la lista de tiques usados en el día actual que se va a mostrar en la clase `UsedTicketListFragment`. La clase extiende de la clase `BaseAdapter` que es un adaptador básico para crear listas.

El atributo más destacable de la clase es:

- `ticketList`: Lista que contiene los elementos que se van a mostrar.

Los métodos más importantes de la clase son:

- `getCount`: Obtiene el número de elementos de la lista.
- `getItem`: Devuelve el elemento de la lista en la posición indicada.
- `getView`: Permite personalizar la vista de cada elemento de la lista.
- `updateTickets`: Permite notificar que la lista ha sufrido cambios y refrescar la vista de la misma.

Clase `TicketListFragment`

Clase que permite mostrar al usuario un listado con los tiques adquiridos usados el día actual. Este fragmento se mostrará en la pantalla principal cuando el usuario seleccione la pestaña «Tickets Usados».

El atributo más destacable de la clase es:

- `ticketList`: Lista que contiene los elementos que se van a mostrar.

El método más importante de la clase es:

- `onListItemClick`: Obtiene los datos del tique seleccionado por el usuario.

Clase `WelcomeScreenActivity`

Clase que muestra la pantalla de inicio y realiza las operaciones necesarias al iniciar la aplicación.

Los atributos más destacables de la clase son:

- `URL`: URL que contiene la IP del servidor y el puerto en el que están disponibles los servicios web.
- `TRUSTSTORE_PASSWORD`: Contraseña del almacén de claves que contiene el certificado de clave pública que permite la comunicación con los servicios web del servidor.

Los métodos más destacados de la clase son:

- `managePIN`: En función de ciertos parámetros permite saber desde que contexto ha sido llamada la clase y ejecuta el método correspondiente.
- `checkConnectivity`: Permite realizar una comprobación de la conectividad a Internet del teléfono móvil antes de intentar cualquier operación que requiera de ella.
- `inputStreamToString`: Permite leer la respuesta recibida como respuesta a la llamada realizada al servicio web.
- `nextScreen`: Comprueba si se ha establecido un código de seguridad y dirige la aplicación a la siguiente pantalla en función de esto.

Dentro de la propia clase se crean varias clases que extienden de `AsyncTask`. Esto permite crear tareas asíncronas que realizan una tarea en segundo plano evitando que la aplicación se bloquee. Estas clases son las que permitirán realizar las llamadas a los servicios web del servidor y son las siguientes:

- `ConsultBlackListTask`: Permite llamar al servicio web que consulta si el usuario está bloqueado.
- `ConsultUsedTicketTask`: Permite llamar al servicio web que devuelve los tiques usados de manera que quedan actualizados en la aplicación.
- `DeleteExpiratedTicketTask`: Permite llamar al servicio web que devuelve los identificadores de los tiques caducados para borrarlos de la aplicación.
- `DeleteSignatureUsedTicketTask`: Permite borrar las firmas de los tiques que no hayan sido usados el día actual.

5.3. Implementación

En esta sección se explicaran en detalle todas las decisiones de diseño tomadas durante la implementación de la aplicación móvil.

5.3.1. Formato de los tiques

Una de las etapas clave durante el diseño e implementación del sistema fue la decisión de establecer un formato estándar de tique definiendo que campos son necesarios para identificar unívocamente un tique así como que campos serían firmados de manera que se pueda verificar tanto la autenticidad como la integridad del tique en cuestión.

Los tiques enviados por el usuario desde el teléfono móvil para acceder, salir o por petición de un revisor del servicio de transporte público siguen el formato que se muestra a continuación:

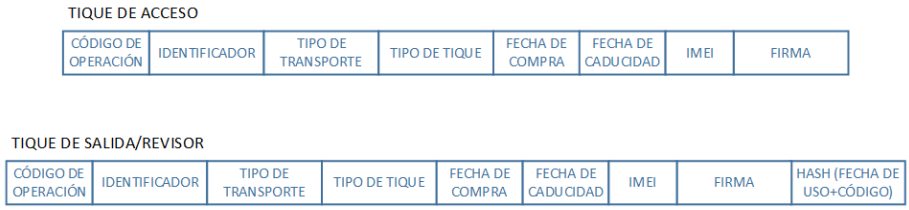

Figura 5.8: Formato de los tiques

- **Código de operación:** Permite al lector saber si el tique está siendo utilizado para acceder o salir del servicio de transporte en función de su valor. El mismo código del lector podría ser utilizado para crear la aplicación que permita validar los tiques a un revisor por lo que también se incluye un valor para esta acción. Este campo vale 0 si el tique se utiliza para acceder, 1 si se utiliza para salir y 2 si se envía un tique a un revisor.
- **Identificador:** Permite identificar un tique unívocamente. Se ha utilizado un identificador universalmente único (UUID, del inglés *Universally Unique IDentifier*) que ofrece unas $3 * 10^{38}$ posibilidades. En su forma canónica, un UUID consiste de 32 dígitos hexadecimales, mostrados en cinco grupos separados por guiones, de la forma 8-4-4-4-12 para un total de 36 caracteres (32 dígitos y 4 guiones). Esta cantidad de combinaciones se ha considerado suficiente para el prototipo pudiendo ser modificado el sistema de identificación si llegará a ser usado en producción y fuera necesario ampliar el número de combinaciones posibles.
- **Tipo de transporte:** Permite conocer el tipo de transporte para el que es válido el tique. En este prototipo se han considerado tres tipos de transporte (metro, cercanías y autobús) siendo posible ampliarlos en un futuro.
- **Tipo de tique:** Permite identificar el tipo de tique. En este prototipo se han considerado dos tipos de tique: sencillo, que permite un único uso y abono mensual, que permite que el usuario acceda durante un mes a

cualquier tipo de transporte. En un futuro podrán ser añadidos nuevos tipos de tique.

- **Fecha de compra:** Fecha en la que el usuario ha adquirido el tique.
- **Fecha de caducidad:** Los tiques tendrán un período de validez desde su compra y este campo indicará la fecha en la que el tique deja de ser válido.
- **Imei:** Permite identificar el dispositivo desde el que se está enviando el tique. Como ya se vio en el apartado del análisis (capítulo 3) el sistema no cuenta con registro de usuarios por lo que los usuarios se identifican con el número de imei de su teléfono. Se enviará el imei para comprobar que el teléfono desde el que se envía el tique es el mismo desde el cual se adquirió.
- **Firma:** Firma de todos los campos anteriores realizada con el certificado del servidor en el momento de su generación. Este campo permite verificar tanto la autenticidad como la integridad del tique.
- **Hash (fecha de uso + código):** En el caso del envío de un tique para salir de un servicio de transporte público, este código se indicará junto al lector y deberá ser introducido por el usuario en la aplicación para generar el tique que le permitirá abandonar el transporte. Este código cambiará cada día de manera que permitirá verificar la frescura en el tique generado. En el caso del envío de un tique a un revisor de un servicio de transporte público, este código será facilitado por el revisor y deberá ser introducido por el usuario en la aplicación para generar el tique. En este caso, cada vez que el revisor solicite un tique se generará aleatoriamente un código.

Estos campos se enviarán como una cadena de texto separando cada uno de ellos con el carácter «]».

5.3.2. Firma digital de los tiques

A la hora de diseñar el sistema siempre ha estado presente la necesidad de que el sistema fuera seguro y pudieran prevenirse en gran medida los posibles ataques a los que estaría expuesto.

En este tipo de sistemas, el principal ataque que puede darse por parte de los usuarios es la falsificación de los tiques. Para evitarlo, todos los tiques emitidos serán firmados en el servidor.

La firma digital es un mecanismo criptográfico que permite al receptor de un mensaje firmado digitalmente determinar la entidad originadora de dicho mensaje (autenticación de origen y no repudio), y confirmar que el mensaje no ha sido alterado desde que fue firmado por el originador (integridad). La firma digital proporciona una herramienta que permite al sistema detectar la falsificación y la manipulación del los tiques por parte de los usuarios.

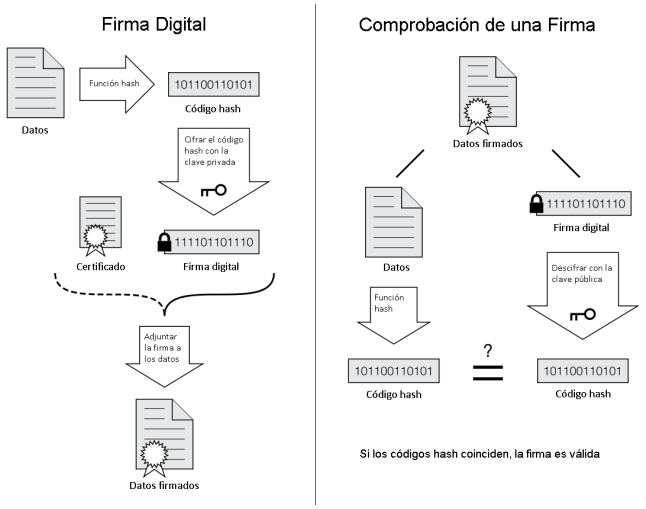

Figura 5.9: Firma digital

Como se muestra en la figura 5.9, el mecanismo de la firma digital consiste en generar en primer lugar con código hash a partir del mensaje original y posteriormente cifrar este código con la clave privada del certificado del emisor. El emisor enviará el mensaje original y la firma digital al receptor. Una vez que el receptor recibe estos datos, procederá a verificar la firma. Para ello el receptor deberá disponer del certificado de clave pública correspondiente al emisor. El receptor generará el código hash a partir del mensaje original y descifrará la firma recibida con la clave pública del certificado de manera que si ambos códigos hash obtenidos son iguales se considera que la firma es válida.

En el caso del sistema propuesto el receptor del mensaje (tique) en primera instancia (el usuario) no necesita validar esta firma y será el lector al que envíe el usuario dicho tique el que dispondrá del certificado de clave pública que le permitirá realizar la validación.

Java cuenta con distintos tipos de algoritmos de firma digital entre los que se encuentran:

- **NONEwithRSA:** No utiliza función de hash sobre los datos antes de cifrarlos utilizando el algoritmo RSA.
- **MD2/MD5withRSA:** Utiliza las funciones de hash MD2 o MD5 respectivamente sobre los datos antes de cifrarlos utilizando el algoritmo RSA tal y como se define en el estándar PKCS#1.
- **SHA1/SHA256/SHA384/SHA512withRSA:** Utiliza las funciones de hash SHA1, SHA256, SHA384 o SHA512 respectivamente sobre los datos antes de cifrarlos utilizando el algoritmo RSA tal y como se define en el estándar PKCS#1.

Para la implementación del sistema se tomó la decisión de utilizar el algoritmo *SHA512withRSA* considerando que el servidor tiene una capacidad de cómputo suficiente para que el coste de operación no sea trascendente. Además,

el único cambio posible sería en el algoritmo de hash y la diferencia de velocidad de cómputo sería significativa si lo único que se hiciera fuera calcular el mismo. Cuando se utiliza el algoritmo de hash en una operación de firma digital, el costo de la operación asimétrica es mucho mayor que la del hash.

Para poder firmar los tiques ha sido necesaria la creación de una clave privada y el certificado de clave pública correspondiente. Para ello se ha utilizado OpenSSL [17].

En primer lugar se ha generado una clave privada RSA de 2048 bits:

```
openssl genrsa -out server.key 2048
```

A continuación se procede a crear el certificado autofirmado que será válido durante 1 año (365 días). Este certificado contiene únicamente la clave pública y será utilizado por la aplicación del lector y el revisor para verificar la firma de los tiques que reciban.

```
openssl req -new -key server.key -out server.csr

openssl x509 -req -days 365 -in server.csr -signkey server.key
-out server.crt
```

Por último se crea el almacén de claves que contendrá tanto la clave privada como el certificado de clave pública en formato PKCS#12. Este almacén de claves solo será utilizado por el servidor para firmar los tiques generados.

```
openssl pkcs12 -export -in server.crt -inkey server.key -out
serverkeystore.p12 -name nfc
```

5.3.3. Cifrado de los tiques en el teléfono móvil

Otro de los posibles ataques a los que se expone este tipo de sistemas es el duplicado de los tiques. Para evitar este tipo de ataque se han tomado varias medidas. La primera, que ya se comento en el apartado de firma de los tiques, es la de incluir el IMEI del dispositivo desde el que se adquiere el tique en los campos sobre los que se produce la firma para evitar que el tique pueda ser copiado a otros dispositivos y usado desde ellos.

Otra de las medidas que se ha tomado ha sido la de cifrar los tiques en el teléfono móvil del usuario. Cuando el usuario adquiere un tique a través del sistema, la firma del tique se cifra antes de almacenarlo en el teléfono móvil y se mantiene cifrada hasta el momento de su uso.

Android dispone de diferentes cifradores de bloque como los que se muestran en la tabla 5.1. La decisión de emplear AES como algoritmo de cifrado simétrico se debe principalmente a que, de todos los cifradores simétricos de bloque que ofrece Android, es el más extendido, es el único que está estandarizado y hoy en día se considera seguro.

Advanced Encrytion Standard (AES) es un algoritmo de cifrado simétrico muy extendido que sigue un esquema de cifrado de bloques. AES, a pesar de ser un algoritmo público y de uso público, está aprobado por la NSA (Agencia de Seguridad Nacional de los EE.UU.) como algoritmo seguro para proteger

información clasificada de nivel SECRETO usando claves de 128 bits y de ALTO SECRETO, si se usan claves de 192 o 256 bits.

Cifrador	**Tamaño de bloque**	**Tamaño de clave (en bits)**
AES	128 bits	0-256
Camellia	128 bits	128, 192, 256
Blowfish	64 bits	0-448
Twofish	128 bits	128, 192, 256

Tabla 5.1: Cifradores de bloque en Android 2.3.3

En Android es posible indicar el modo de operación que va a emplear el cifrador de bloque. El modo por defecto es *Electronic Code Book* (ECB) que es el modo más sencillo ya que un bloque de texto plano da lugar a un bloque de texto cifrado. Este modo no ofrece protección contra un análisis criptográfico de reconocimiento de patrones. Para prevenir este tipo de ataques se ha cambiado el modo de operación a *Cipher-Block Chaining* (CBC) que utiliza un valor adicional conocido como vector de inicialización (IV) que se utiliza para realizar la operación XOR sobre el primer bloque de texto plano.

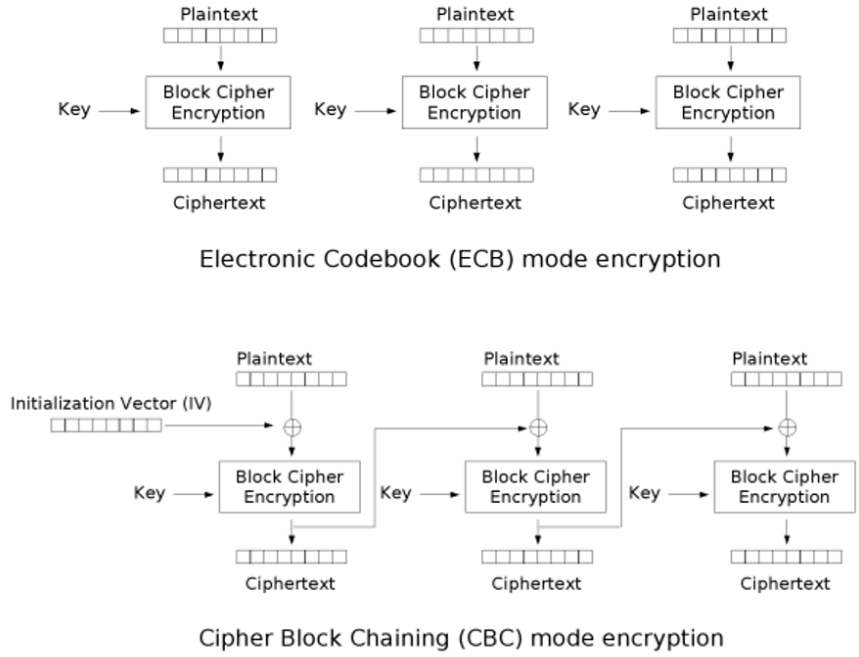

Figura 5.10: Modos de operación

Otro factor a tener en cuenta en los cifradores de bloque es el mecanismo de relleno. Estos cifradores trabajan con un tamaño de bloque fijo y en la mayoría de los casos el último bloque de texto plano a cifrar es más pequeño que dicho tamaño por lo que hace falta rellenar con caracteres los huecos restantes hasta

completarlo. Por defecto se suele emplear el llamado *Zero Padding* que consiste en rellenar con ceros los huecos que faltan. En el caso del sistema se va a emplear el *PKCS#5 Padding* que toma el número de huecos que faltan por rellenar y lo usa como bit de relleno.

La principal desventaja del cifrado simétrico es la necesidad de conservar la clave para poder descifrar. Si la clave se almacena junto con los datos cifrados o en los archivos privados de la aplicación es bastante sencillo extraerla, expecialmente en un teléfono rooteado[1], y descifrar los datos. Android ofrece dos posibles soluciones a este problema: proteger la clave con un servicio del sistema o no almacenar la clave. Para la primera opción, desde su versión 4.0 Android proporciona un servicio que es accesible desde la clase `KeyChain`. Sin embargo, por ahora solo se puede utilizar para almacenar claves privadas RSA y certificados. En el segundo caso, no almacenar la clave, hay que tener en cuenta que las claves simétricas son cadenas de bits aleatorios de un tamaño considerable como para ser recordadas. Para facilitar la tarea existen técnicas de derivación de claves criptográficas fuertes a partir de una contraseña que puede ser fácilmente recordada por un usuario.

Se han considerado las principales opciones que ofrece Android para utilizar *Password-based Encryption* (PBE): `PBKDF2WithHmacSHA1` (definido en PKCS#5) y `PBEWITHSHA256AND256BITAES-CBC-BC` (definido en PKCS#12). Ambas técnicas consisten realizar múltiples iteraciones sobre un hash formado por la contraseña introducida por el usuario y un salt aleatorio. El tamaño del salt y el número de iteraciones que se realizan aumentan la complejidad de la operación para generar la clave final.

Se decidió utilizar `PBEWITHSHA256AND256BITAES-CBC-BC` que permite derivar una clave AES de 256 bits para cifrar utilizando SHA256 como algoritmo de hash al realizar la derivación de la clave. El motivo de esta decisión es principalmente el uso de SHA256 como función de hash en lugar de SHA1 (que es el algoritmo que emplea la función de derivación de clave PBKDF2) ya que sobre este último se han realizado ataques que han comprometido su seguridad y el primero pertenece a la familia SHA2, sucesora de SHA1. Otro de los motivos, es que la derivación de la clave empleando el algoritmo `PBEWITHSHA256AND256BITAES-CBC-BC` es más rápido que con el algoritmo `PBKDF2WithHmacSHA1` tal y como se muestra en la figura 5.11, dato a tener en cuenta si consideramos que se realizaran 5000 iteraciones que hacen este cálculo para derivar la clave final con la que se cifrarán los datos.

El número de iteraciones y el tamaño del salt se han estimado en base al dato de como cifra Android sus *backups*: genera una clave AES de 256 bits utilizando 10000 iteraciones con un salt de 512 bits. La clave utilizada por la aplicación para cifrar se generará realizando 5000 iteraciones con un salt de 128 bits.

[1]Rootear el teléfono es un proceso que permite obtener privilegios de superusuario o *root* en el mismo.

Figura 5.11: Tiempos de derivación de clave en un Samsung Galaxy S3

En el sistema propuesto, cuando el usuario inicia por primera vez la aplicación se le pedirá que establezca un código de seguridad de 4 dígitos. Una vez establecido este código se genera una clave aleatoria AES de 256 bits con la que se cifraran las firmas de los tiques adquiridos. A su vez, esta clave se cifrará con la clave que se derivará del código de seguridad elegido por el usuario de forma que para poder usar un tique es necesario conocer dicho código. Cada tique se cifrara utilizando un vector de inicialización diferente que se almacenará junto con el texto cifrado.

El motivo de este diseño se debe al coste (en tiempo) que conllevan las operaciones criptográficas conociendo las limitaciones que supone que se ejecuten en teléfono móvil. Si se cifrarán los tiques con la clave derivada y el usuario deseará cambiar su código de seguridad, opción que se ha considerado indispensable, el coste de este cambio dependería en gran medida del número de tiques almacenados en el dispositivo ya que habría que descifrarlos todos con la clave derivada antigua y volverlos a cifrar con la nueva clave derivada. Con el sistema propuesto, este cambio únicamente supondría descifrar la clave AES con la que se cifran los tiques con la clave derivada antigua y cifrarla de nuevo con la nueva clave derivada.

5.3.4. Almacenamiento de los tiques en el teléfono móvil

Como ya se comento en el capítulo 3 de análisis, es necesario almacenar de manera persistente los tiques adquiridos por el usuario en el dispositivo. Android dispone de cuatro mecanismos que permiten almacenar datos de manera persistente en el dispositivo.

- **Shared Preferences:** Permite almacenar tipos de datos primitivos (int, Boolean, float, long y String) de manera persistente en un fichero XML. Tiene cuatro modos de privacidad (MODE_PRIVATE, MODE_WORLD_READABLE, MODE_WORLD_WRITABLE y MODE_MULTI_PROCESS) que limitan el acceso a dicho fichero.

- **Almacenamiento interno:** Permite almacenar datos en la memoria interna del teléfono. Generalmente estos datos no son accesibles ni por otras aplicaciones ni por el usuario. Cuando la aplicación se elimina del dispositivo, los datos almacenados en la memoria interna por la aplicación también son borrados. Tiene tres modos de privacidad (MODE_PRIVATE, MODE_WORLD_READABLE y MODE_WORLD_WRITABLE) que limitan el acceso a los datos.

- **Almacenamiento externo:** Permite almacenar datos en la memoria externa del teléfono (normalmente tarjetas SD). Estos datos son totalmente accesibles por otras aplicaciones por el usuario. **Base de datos SQLite:** Permite la aplicación crear una base datos en el teléfono. Las bases de datos creadas por una aplicación únicamente son accesibles desde las clases de la propia aplicación. Cuando la aplicación se elimina del dispositivo, la base de datos creada por la aplicación y todo su contenido también se borra.

Tras estudiar las distintas posibilidades se decidió emplear el almacenamiento a través de bases de datos SQLite por ser la opción se mejor se ajusta a los requisitos de la aplicación.

SQLite es un motor de bases de datos muy popular en la actualidad por características como su pequeño tamaño, no necesitar servidor, precisar poca configuración, ser transaccional y ser de código libre.

Android incorpora de serie todas las herramientas necesarias para la creación y gestión de bases de datos SQLite, y entre ellas, una completa API para llevar a cabo de manera sencilla todas las tareas necesarias. En Android, la forma típica para crear, actualizar, y conectar con una base de datos SQLite es a través de una clase auxiliar llamada `SQLiteOpenHelper` o una clase propia que derive de ella y que se puede personalizar para que se adapte a las necesidades concretas de la aplicación a desarrollar. La clase `SQLiteOpenHelper` tiene tan sólo un constructor, que normalmente no es necesario sobrescribir, y dos métodos abstractos, `onCreate()` y `onUpgrade()`, que se personalizan con el código necesario para crear la base de datos de la aplicación y para actualizar su estructura respectivamente.

Modelo de datos

El esquema del modelo de datos empleado se muestra en la figura 5.12 y es muy simple: la base de datos contiene una tabla `tickets` con siete columnas que permiten almacenar toda la información necesaria de los tiques adquiridos por el usuario.

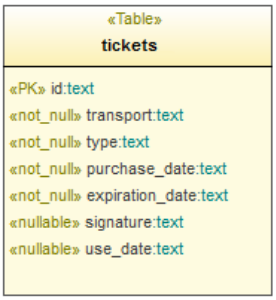

Figura 5.12: Modelo de la base de datos

- **id (identificador):** Permite identificar un tique unívocamente.
- **transport (tipo de transporte):** Permite conocer el tipo de transporte para el que es válido el tique.
- **type (tipo de tique):** Permite identificar el tipo de tique.
- **purchase_date (fecha de compra):** Fecha en la que el usuario ha adquirido el tique.
- **expiration_date (fecha de caducidad):** Los tiques tendrán un período de validez desde su compra y este campo indicará la fecha en la que el tique deja de ser válido.
- **signature (firma):** Firma de ciertos campos del tique realizada con el certificado del servidor en el momento de su generación.
- **use_date (fecha de uso):** Fecha en la que el tique fue usado por el usuario.

Los únicos campos que pueden tener un valor nulo son la fecha de uso ya que será nulo hasta el momento en el que el usuario utilice dicho tique y la firma, que contendrá la firma hasta que el usuario utilice un tique para salir de un servicio de transporte público o la fecha de uso sea anterior a la fecha actual. La medida de borrar la firma se adopta para prevenir la duplicación de tiques ya usados por el usuario.

5.3.5. Firma de la aplicación

Para que una aplicación pueda ser instalada en un dispositivo Android es obligatorio que haya sido firmada con un certificado cuya clave privada sea propiedad del desarrollador de la aplicación. El certificado sirve como medio para identificar al desarrollador de una aplicación y establecer relaciones de confianza entre aplicaciones. A la hora de firmar una aplicación Android hay varios puntos a tener en cuenta:

- Android no permitirá la instalación de aplicaciones no firmadas por lo que la firma de la aplicación es imprescindible.
- Android permite firmar aplicaciones con certificados autofirmados.

- Android solo comprueba la validez del certificado en el momento de la instalación de la aplicación por lo que si el certificado expira una vez instalada la aplicación no afectará a su funcionamiento.

- Se pueden utilizar herramientas como Keytool o Jarsigner para firmar aplicaciones una vez empaquetadas en un archivo con la extensión `.apk`.

En el caso de este proyecto, la firma de aplicaciones incrementa la seguridad de la aplicación móvil desarrollada ya que evita que desarrolladores maliciosos modifiquen la misma y la publiquen como legítima. Un desarrollador malicioso no puede publicar una aplicación modificada en *Google Play*. Si se diera el caso, la aplicación estaría firmada la clave privada del desarrollador malicioso de manera que sería reconocido como el autor de la aplicación modificada eximiendo de cualquier problema al desarrollador legítimo de la aplicación.

Una vez firmado el archivo `.apk` de la aplicación, Google recomienda utilizar la herramienta *Zipalign* sobre este archivo. *Zipalign* es una herramienta que viene incluida en los SDK de Android desde Donut 1.6 y que está pensada para optimizar los paquetes `.apk` adaptándolos a los requisitos óptimos del sistema Android.

En Android, los datos almacenados dentro de archivos `.apk` son requeridos por multitud de procesos: el instalador leerá el manifiesto para manejar los permisos asociados con cada solicitud; la aplicación Inicio leerá los recursos para obtener el nombre de la aplicación y el icono; el servidor del sistema leerá los recursos por diversos motivos (p.ej. para mostrar notificaciones); y por supuesto los archivos de recursos son obviamente utilizados por la propia aplicación.

Zipalign garantiza que todos los datos sin comprimir empiezan con una particular alineación de bytes respecto al comienzo del archivo. Establecer una alineación de 4 bytes proporciona una optimización de rendimiento cuando se instala en un dispositivo Android. Cuando están alineados, el sistema es capaz de leer archivos con `mmap()`, incluso si contienen datos binarios con restricciones de alineamiento, en vez de copiar todos los datos del paquete en el caso de no estar alineados.

Si una aplicación no está optimizada con *Zipalign* la lectura de los recursos de aplicaciones será lento y requerirá de mucha memoria. En el mejor de los casos, el único resultado visible es que tanto la aplicación principal como el inicio de la aplicación será más lenta de lo que deberían. En el peor de los casos, la instalación de varias aplicaciones no alineadas aumentará los requisitos de memoria, provocando que el sistema se sobrecargue por tener que iniciar y terminar estos procesos. En estos casos el usuario terminará con un dispositivo lento y con un consumo de batería excesivo.

5.3.6. Ofuscación del código de la aplicación

ProGuard es una librería desarrollada en Java que permite optimizar y ofuscar el código Java una vez que este está compilado. Una de las principales utilidades de *ProGuard* es que elimina todas las clases no usadas del archivo `.apk`. Es muy útil si la aplicación hace uso de librerías externas, ya que es bastante usual utilizar únicamente una parte de la librería dejando multitud de

clases inútiles ocupando espacio. *ProGuard* detectará estas clases y las eliminará del archivo resultante, disminuyendo bastante el tamaño de la aplicación en determinados casos.

Por otro lado, optimiza el *bytecode* de Java ofuscando el código. Esto es, cambiando el nombre de las variables, métodos y clases, haciendo que ocupen mucho menos espacio y dejando el código incomprensible para aquellos que intenten usar un decompilador sobre la aplicación.

5.4. Interfaces de usuario

Una de las fases más importantes durante la implementación de cualquier aplicación que deba interactuar con un usuario, especialmente si se trata de una aplicación móvil, es su interfaz. Se ha comprobado en múltiples ocasiones que los usuarios prefieren aplicaciones que sean sencillas de utilizar con interfaces de usuario intuitivas. Por este motivo muchas aplicaciones potentes y con buenas características no han tenido el éxito esperado.

A la hora de realizar el diseño de las interfaces es importante conocer las funcionalidades que ofrece la aplicación para organizarlas de forma que simplifique su empleo al usuario. La aplicación consta de tres funcionalidades principales: adquirir tiques, utilizar tiques para acceder al servicio de transporte público y utilizar tiques usados para salir del mismo o mostrarlo al revisor. Se decidió crear una pantalla principal divida en pestañas que permita al usuario elegir entre estas tres opciones seleccionando directamente la pestaña deseada o deslizando el dedo hacía los lados para moverse.

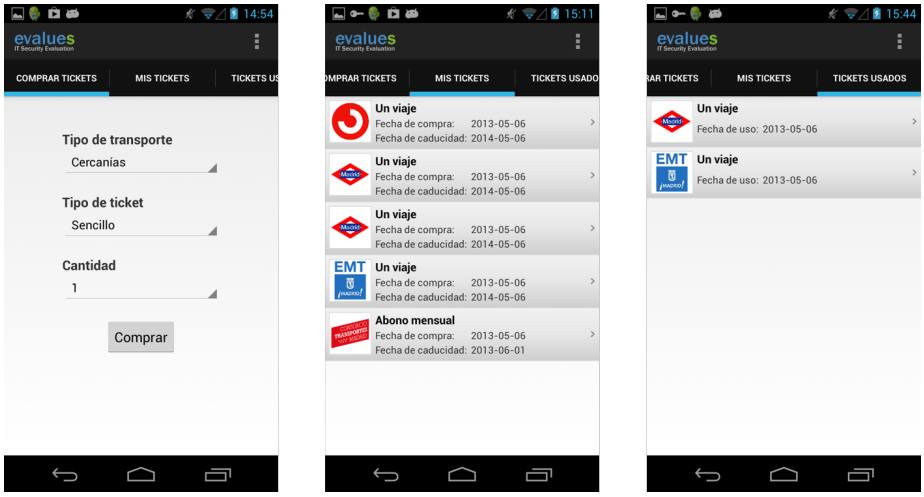

Figura 5.13: Interfaces de la pantalla principal

La aplicación cuenta con una pantalla de inicio (figura 5.14) que da paso a la pantalla principal una vez que se realiza la sincronización inicial con el servidor.

Figura 5.14: Interfaz de bienvenida

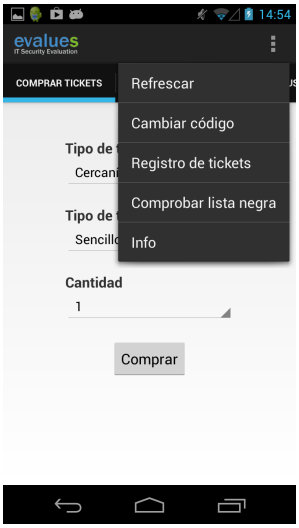

Figura 5.15: Interfaz del menú de opciones

La aplicación es capaz de realizar otras acciones como refrescar los tiques, cambiar el código de seguridad, mostrar el registro de todos los tiques adquiridos por el usuario o consultar la lista negra, que serán accesibles desde el menú de opciones de la aplicación como se muestra en la figura 5.15.

En la figura 5.16 se muestra la interfaz de la pantalla del registro donde aparecen todos los tiques adquiridos por el usuario. El usuario puede elegir entre mostrar todos, mostrar solo los no usados o únicamente los usados.

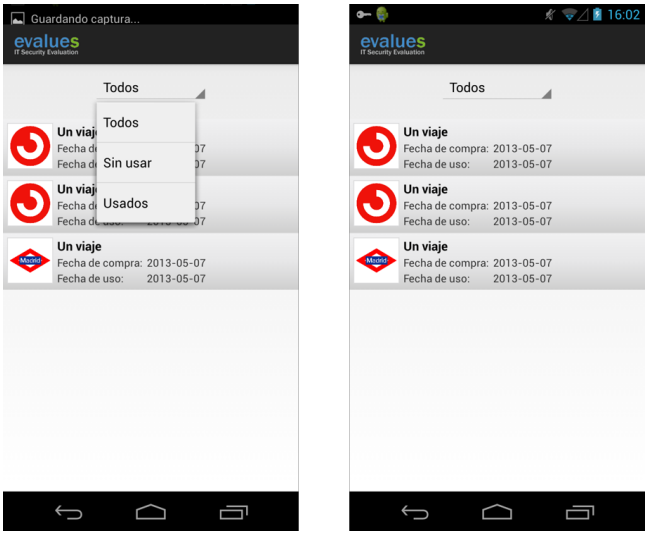

Figura 5.16: Interfaces del registro

Finalmente se muestra la interfaz de introducción de código de seguridad (figura 5.17) que aparecerá tanto para establecer el código como para que el

usuario introduzca el código establecido.

Figura 5.17: Interfaz de introducción de código de seguridad

Para diseñar estas interfaces se han intentado utilizar interfaces simples, homogénea y que mantengan los criterios de usabilidad a los que un usuario de la plataforma Android está acostumbrado. Algunas de las directrices que se han seguido para diseñar las interfaces de usuario de la aplicación son las siguientes:

- Mantener siempre un acceso rápido a los contenidos.
- Las interfaces seguirán un aspecto homogéneo de forma que se mantenga un conjunto coherente de colores, fuentes y estilos para la aplicación.
- Posición y tamaño de los botones en función de su frecuencia de uso.
- Mantener las convenciones a las que está acostumbrado un usuario de la plataforma Android.
- Seguir las indicaciones presentadas por Google en la página oficial de desarrolladores de Android en lo que a diseño de iconos e interfaces se refiere.

6

Diseño del servidor

6.1. Diagrama de clases

En las siguientes secciones se describirá el diseño y la implementación de la aplicación del servidor que contiene los servicios web que pueden ser accedidos por las aplicaciones clientes. La figura 6.1 muestra el diagrama de clases de la aplicación con los métodos más relevantes.

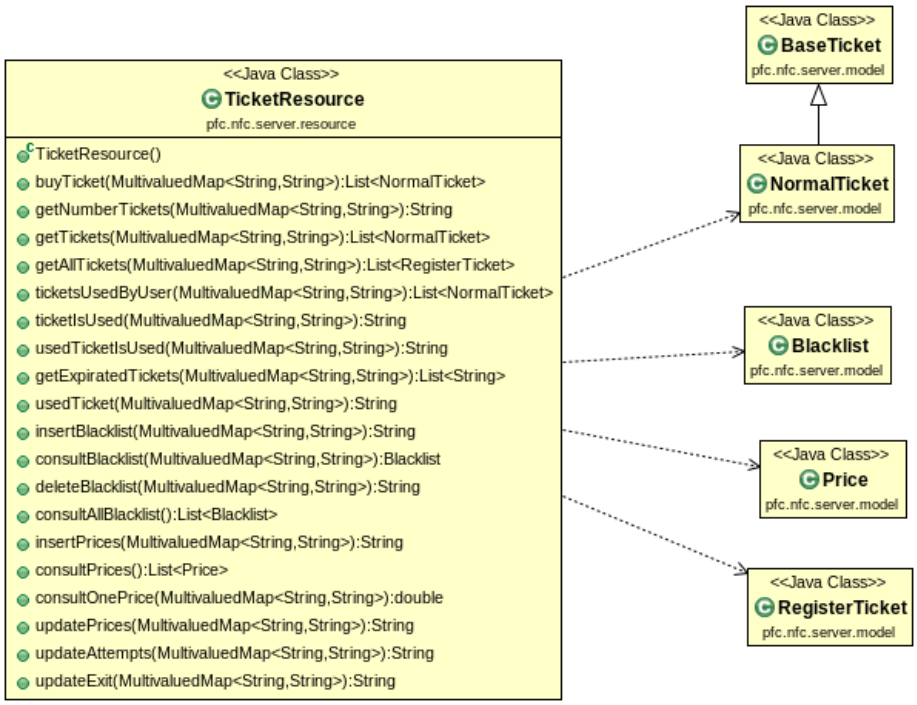

Figura 6.1: Diagrama de clases de la aplicación del servidor

6.2. Definición de las clases

En el siguiente apartado se va a explicar la función de cada clase que aparece en el diagrama comentando los atributos y métodos más relevantes.

Clase `TicketResource`

Clase principal de la aplicación que muestra implementa los métodos a que serán llamados a través de servicios web por las aplicaciones cliente. Los métodos más destacados son:

- `buyTicket`: Implementa la funcionalidad del servicio web que permite al usuario adquirir tiques desde la aplicación móvil instalada en su dispositivo. La aplicación envía el tipo de transporte, el tipo de tique, el número de tiques y el imei del teléfono desde el que se realiza la compra; el método genera los tiques, los firma y los almacena en la base de datos para finalmente devolver una lista con todos ellos a la aplicación que realizó la petición.

- `getTickets`: Implementa la funcionalidad del servicio web que permite a la aplicación móvil actualizar los tiques cuando el usuario selecciona la opción refrescar. La aplicación envía el imei del teléfono y el método consulta la base de datos para devolver una lista con los todos los tiques no caducados adquiridos por el usuario.

- `getAllTickets`: Implementa la funcionalidad del servicio web que permite a la aplicación móvil mostrar todos los tiques adquiridos cuando el usuario selecciona la opción registro. La aplicación envía el imei del teléfono y el método consulta la base de datos para devolver una lista con los todos los tiques adquiridos por el usuario.

- `ticketsUsedByUser`: Implementa la funcionalidad del servicio web que permite a la aplicación móvil actualizar los tiques usados durante el inicio. La aplicación envía el imei del teléfono y el método consulta la base de datos para devolver una lista con los todos los tiques usados no caducados utilizados por el usuario.

- `ticketIsUsed`: Implementa la funcionalidad del servicio web que permite a la aplicación lector comprobar si el tique utilizado por el usuario para acceder al servicio de transporte público ha sido utilizado previamente. La aplicación envía el identificador del tique y el método consulta en la base de datos si el tique ha sido usado devolviendo el resultado.

- `usedTicketIsUsed`: Implementa la funcionalidad del servicio web que permite a la aplicación lector comprobar si el tique utilizado por el usuario para salir del servicio de transporte público ha sido utilizado previamente. La aplicación envía el identificador del tique y el método consulta en la base de datos si el tique ha sido usado devolviendo el resultado.

- `getExpiratedTickets`: Implementa la funcionalidad del servicio web que permite a la aplicación móvil eliminar los tiques caducados del teléfono

correspondiente. La aplicación envía el imei del teléfono y el método consulta la base de datos para devolver una lista con los identificadores de los tiques caducados.

- `usedTicket`: Implementa la funcionalidad del servicio web que permite a la aplicación del lector actualizar la fecha de uso. La aplicación envía el imei del teléfono y el método consulta la base de datos para devolver una lista con los identificadores de los tiques caducados.

- `insertBlacklist`: Implementa la funcionalidad del servicio web que permite a la aplicación móvil insertar a un usuario en la lista negra del sistema por alcanzar el número máximo de intentos de introducción del código de seguridad. La aplicación envía el imei del teléfono y el método lo inserta en la base de datos junto con la fecha y el motivo de la inserción.

- `consultAllBlacklist`: Implementa la funcionalidad del servicio web que permite a la aplicación del lector actualizar periódicamente la lista de usuarios incluidos en la lista negra del sistema por hacer intentar hacer un uso fraudulento de la aplicación. La aplicación llama al método y este consulta la base de datos para devolver una lista con los números de imei incluidos en la lista negra.

- `consultOnePrice`: Implementa la funcionalidad del servicio web que permite a la aplicación móvil consultar el precio de un tipo de tique y mostrar al usuario el total que debe pagar antes de confirmar la compra. La aplicación envía el tipo de transporte y el tipo de tique, y el método consulta la base de datos para devolver el precio correspondiente.

- `updateAttempts`: Implementa la funcionalidad del servicio web que permite a la aplicación del lector actualizar el número de intentos fraudulentos de un usuario. La aplicación envía el imei y este actualiza el número de intentos de uso fraudulento del usuario indicado.

- `updateExit`: Implementa la funcionalidad del servicio web que permite a la aplicación del lector actualizar el estado del tique indicando si ya ha sido utilizado para salir del servicio de transporte público. La aplicación envía el identificador del tique y este actualiza el estado.

Contiene el método `checkConnectivity` que permite realizar una comprobación de la conectividad a Internet del teléfono móvil antes de intentar cualquier operación que requiera de ella.

Clases `BaseTicket, Blacklist, NormalTicket, Price, RegisterTicket, Transport, Type`

Estas clases son representaciones de objetos que pueden ser serializados fácilmente en objetos JSON que formaran las respuestas que devolverá el servidor al consumidor de sus servicios web.

6.3. Implementación

6.3.1. REST VS SOAP

Durante el diseño de la aplicación se consideró que ni la aplicación móvil ni la aplicación del lector deberán acceder directamente a la base de datos por lo que se decidió implementar servicios web que realizan esta tarea.

Un servicio web es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos. Para implementar estos servicios web se tuvieron en cuenta dos posibilidades: SOAP (*Simple Object Access Protocol*) y REST (*Representational State Transfer*).

SOAP es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. Este protocolo deriva de un protocolo creado por David Winer en 1998, llamado XML-RPC. SOAP fue creado por Microsoft, IBM y otros y está actualmente bajo el auspicio de la W3C. Este protocolo basado en XML consiste de tres partes: un sobre (*envelope*), el cual define qué hay en el mensaje y cómo procesarlo; un conjunto de reglas de codificación para expresar instancias de tipos de datos; y una convención para representar llamadas a procedimientos y respuestas.

Por otro lado, REST es una técnica de arquitectura software para sistemas hipermedia distribuidos como la *World Wide Web*. El término se originó en el año 2000, en una tesis doctoral sobre la web escrita por Roy Fielding, uno de los principales autores de la especificación del protocolo HTTP y ha pasado a ser ampliamente utilizado por la comunidad de desarrollo. Un concepto importante en REST es la existencia de recursos (elementos de información), que pueden ser accedidos utilizando un identificador global (un Identificador Uniforme de Recurso). Para manipular estos recursos, los componentes de la red (clientes y servidores) se comunican a través de una interfaz estándar (HTTP) e intercambian representaciones de estos recursos. La petición puede ser transmitida por cualquier número de conectores (por ejemplo clientes, servidores, cachés, túneles, etc.) pero cada uno lo hace sin "ver más allá"de su propia petición (sin estado).

Se realizó una comparativa entre estas dos teconologías considerando las ventajas e inconvenientes de cada una de ellas teniendo siempre presentes las necesidades de la aplicación. En este caso no es necesario que los servicios web tengan estado y tanto el productor como el consumidor conocen el contexto y contenido que van a intercambiar. También hay que tener en cuenta que el principal consumidor de los servicios web será un teléfono móvil que cuenta con unos recursos más limitados.

Finalmente se tomó la decisión de utilizar servicios web que utilicen una arquitectura REST ya que son los que más se ajustan a las necesidades del sistema que se ha implementado. Las ventajas de REST derivan de su simplicidad entre las que se pueden destacar mejores tiempos de respuesta y disminución de sobrecarga tanto en cliente como en servidor, mayor estabilidad frente a futuros

cambios y una gran sencillez en el desarrollo de clientes, ya que estos solo han de ser capaces de realizar interacciones HTTP y codificar información en XML o JSON.

6.3.2. Hypertext Transfer Protocol Secure (HTTPS)

El servidor del sistema implementa una serie de servicios web que ofrecen gran parte de la funcionalidad global por lo que se ha considerado conveniente que el tráfico entre los consumidores de estos servicios y el propio servidor no sea vulnerable a un ataque *man in the middle*. Para proteger el tráfico se han configurado los servicios de manera que los consumidores únicamente puedan comunicarse con el servidor usando el protocolo HTTPS.

Figura 6.2: HTTP VS HTTPS

Al usar HTTPS se adopta una codificación con certificado digital SSL que permite crear un canal de comunicación seguro para el tráfico de datos entre un cliente y el servidor. HTTPS no evita que terceros puedan observar las comunicaciones sino que crea un sistema de cifrado que hace que el mensaje solo pueda ser entendido por el destinatario. Además de establecer un canal de comunicación seguro, HTTPS permite realizar una autenticación de uno o ambos extremos de la comunicación garantizando que el receptor de los datos es quien dice ser. En el caso del sistema, permite a la aplicación cliente saber que se está comunicando con el servidor correcto.

Para poder establecer la comunicación utilizando el protocolo HTTPS ha sido necesaria la creación de una clave privada y el certificado de clave pública correspondiente. Para ello se ha utilizado OpenSSL [17].

En primer lugar se ha generado una clave privada RSA de 2048 bits:

```
openssl genrsa -out tomcat.key 2048
```

A continuación se procede a crear el certificado autofirmado que será válido durante 1 año (365 días):

```
openssl req -new -key server.key -out server.csr

openssl x509 -req -days 365 -in server.csr -signkey server.key
-out server.crt
```

Seguidamente se crea el almacén de claves que contendrá tanto la clave privada como el certificado de clave pública en formato PKCS#12:

```
openssl pkcs12 -export -in server.crt -inkey server.key -out
serverkeystore.p12 -name nfc
```

Por último se importa el certificado de clave pública a un almacén de claves en formato de JKS (Java KeyStore) que es formato de Java. Este almacén de claves será por las aplicaciones cliente (aplicación móvil y lector) para realizar las llamadas a los servicios web disponibles en el servidor.

```
keytool -import -file tomcat.crt -keystore tomcat.jks -keypass
tomcat
```

Android no trabaja directamente con almacenes de claves en formato JKS por lo que hay que convertir en formato BKS (*Bouncy Castle KeyStore*). Bouncy Castle [18] es una librería de rutinas criptográficas que Android utiliza en la mayoría de sus implementaciones de criptografía, incluidos los almacenes de claves. Para poder realizar esta conversión ha sido necesario descargar la librería `bcprov-jdk15on-1.46.jar` desde el sitio web de Bouncy Castle. Una vez descargada se han ejecutado los siguientes comandos para realizar la conversión:

```
keytool -export -alias nfc -keystore tomcat.jks -storepass tomcat
-keypass tomcat -file cert.cer

keytool -import -file cert.cer -keypass 1234 -keystore phonebks.bks
-storetype BKS -storepass 123456 -providerClass
org.bouncycastle.jce.provider.BouncyCastleProvider -providerpath
bcprov-jdk15on-1.46.jar -alias tomcat
```

HTTPS en el servidor Tomcat 7

Para habilitar el acceso a los servicios web a través de HTTPS en el servidor ha sido necesario seguir los siguientes pasos:

1. Se copia el almacén de claves en la localización deseada:
   `cp [origen]/tomcatkeystore.p12 /opt/tomcat/certificate/`

2. Se configura el conector en el archivo *$CATALINA_HOME/conf/server.xml* de la siguiente manera:

```
<!-- Define a SSL Coyote HTTP/1.1 Connector on port 8443 -->
<Connector
       protocol="HTTP/1.1"
       port="8443" maxThreads="200"
       scheme="https" secure="true" SSLEnabled="true"
       keystoreFile="/opt/tomcat/certificate/tomcatkeystore.p12"
       keystorePass="tomcat" keystoreType="PKCS12"
       clientAuth="false" SSLProtocol="TLS"/>
```

HTTPS en los servicios web

Para configurar los servicios web de manera que solo puedan ser utilizados a través de HTTPS se modifica el fichero *WebContent/WEB-INF/web.xml* que se encuentra en la carpeta del proyecto donde se han implementado los servicios web de la siguiente manera:

```
<security-constraint>
        <web-resource-collection>
                <web-resource-name>SSL Forward</web-resource-name>
                <url-pattern>/*</url-pattern>
        </web-resource-collection>
        <user-data-constraint>
                <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
</security-constraint>
```

6.3.3. Servicios web en Java con Jersey

Una vez decidido como se implementarían los servicios web se consideró como implementarlos. Para poder desplegar los servicios web en el servidor Apache Tomcat los servicios web debían estar implementados en Java por lo que se empleó Jersey para realizar la implementación.

Java define el soporte REST para implementar servicios en la *Java Specification Request* (JSR) 311. Esta especificación es conocida como JAX-RS (*Java API for RESTful Web Services*). JAX-RS (*Java API for RESTful Web Services*) es una API del lenguaje de programación Java que proporciona soporte en la creación de servicios web de acuerdo con el estilo arquitectónico *Representational State Transfer* (REST). JAX-RS proporciona algunas anotaciones para ayudar a mapear una clase recurso (un POJO) como un recurso web. Entre estas anotaciones se incluyen:

- @Path especifica la ruta de acceso relativa para una clase recurso o método.
- @GET, @PUT, @POST, @DELETE y @HEAD especifican el tipo de petición HTTP de un recurso.
- @Produces especifica los tipos de medios MIME de respuesta.
- @Consumes especifica los tipos de medios de petición aceptados.

Jersey es la implementación de referencia de Oracle de JAX-RS. Jersey básicamente contiene un servidor REST y un cliente REST. El núcleo del cliente proporciona una librería que permite comunicarse con el servidor.

En el lado del servidor Jersey utiliza un servlet que escanea clases predefinidas para identificar recursos REST. A través del archivo de de configuración `web.xml` de la aplicación web se registra el servlet proporcionado por Jersey. La URL base del servlet es:

`http://dominio_o_IP:puerto/nombre_servicios/patrón_url/ruta_método`

El servlet analiza las peticiones HTTP entrantes y selecciona la clase y el método correctos para responder la la petición. Esta selección se basa en las anotaciones de las clases y los métodos.

6.3.4. Almacenamiento de la información en el servidor

Como ya se comento en el capítulo 3 de análisis, el servidor ejecutará un motor de bases de datos que permitirá almacenar de manera persistente toda la información relevante para la aplicación. Se consideraron dos opciones de software libre como MySQL y PostgreSQL.

MySQL es un sistema de gestión de bases de datos relacional, multihilo y multiusuario muy popular. MySQL se desarrolla como software libre en un esquema de licenciamiento dual. Por un lado se ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia, pero para aquellas empresas que quieran incorporarlo en productos privativos deben comprar a la empresa una licencia específica que les permita este uso. Está desarrollado en su mayor parte en ANSI C. Algunos de los puntos fuertes de son su velocidad a la hora de realizar las operaciones y su bajo consumo que lo hacen apto para ser ejecutado en una máquina con escasos recursos.

PostgreSQL es un SGBD relacional orientado a objetos y libre, publicado bajo la licencia BSD. Esta licencia tiene menos restricciones en comparación con otras como la GPL estando muy cercana al dominio público. Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una empresa y/o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de forma desinteresada, altruista, libre y/o apoyados por organizaciones comerciales. Dicha comunidad es denominada el PGDG (*PostgreSQL Global Development Group*). PostgreSQL posee una gran escalabilidad. Es capaz de ajustarse al número de CPUs y a la cantidad de memoria que posee el sistema de forma óptima, haciéndole capaz de soportar una mayor cantidad de peticiones simultáneas de manera correcta. Además permite la declaración de funciones propias, así como la definición de disparadores.

Se realizó una valoración teniendo en cuenta las necesidades del sistema. A pesar de ser un prototipo, el sistema está pensado para recibir miles de llamadas al día con sus respectivos accesos a la base de datos por lo que el motor de bases de datos seleccionado debe ser capaz de gestionar bases de datos realmente grandes con una rápida velocidad de respuesta. Además, la base de datos debe ser capaz ejecutar disparadores sin reducir de forma significativa el rendimiento de la misma.

Finalmente se optó por instalar el motor de bases de datos PostgreSQL ya que se se adapta mejor en ambientes con altas cargas de usuario y consultas complejas donde la integridad de los datos es muy importante.

Modelo de datos

El diseño de la base de datos se muestra a continuación:

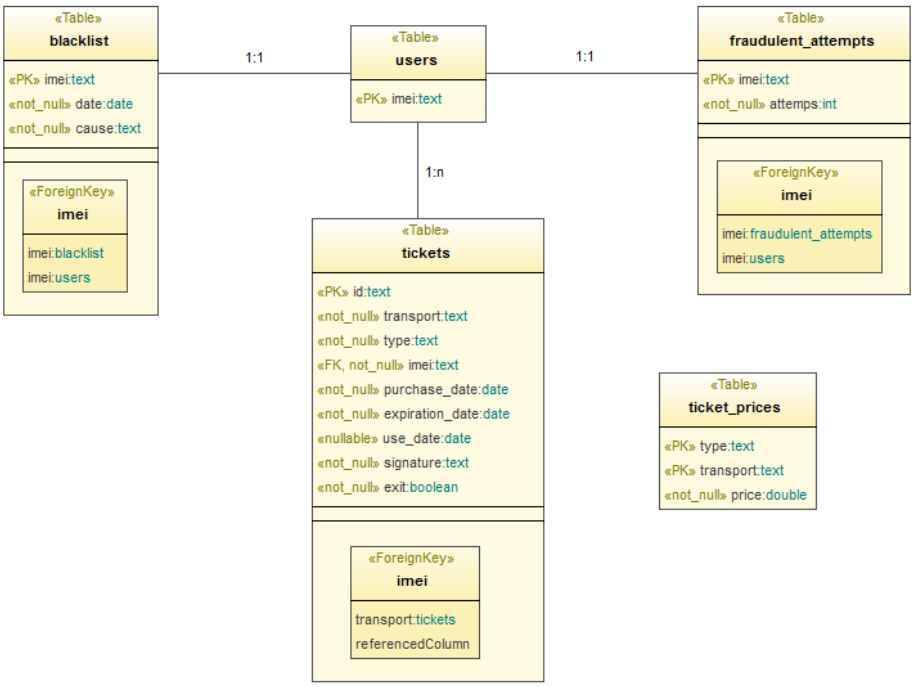

Figura 6.3: Modelo de la base de datos

La base de datos consta de cinco tablas que almacenan toda la información que necesita la aplicación para funcionar correctamente.

Tabla `tickets`

- **id (identificador):** Permite identificar un tique unívocamente.
- **imei:** Permite identificar al usuario que ha adquirido el tique.
- **purchase_date (fecha de compra):** Fecha en la que el usuario ha adquirido el tique.
- **expiration_date (fecha de caducidad):** Los tiques tendrán un período de validez desde su compra y este campo indicará la fecha en la que el tique deja de ser válido.
- **use_date (fecha de uso):** Fecha en la que el tique fue usado por el usuario.
- **signature (firma):** Firma de ciertos campos del tique realizada con el certificado del servidor en el momento de su generación.
- **exit:** Campo que permite saber si el usuario ha usado el tique para salir del servicio de transporte público.

Tabla `users`

- **imei:** Permite identificar a los usuarios que han interactuado con el sistema.

Tabla `blacklist`

- **imei:** Permite identificar al usuario que ha sido incluido en la lista negra.
- **date:** Fecha en la que el usuario ha sido incluido en la lista negra.
- **cause:** Motivo por el cual el usuario ha sido incluido en la lista negra.

Tabla `fraudulent_attempts`

- **imei:** Permite identificar al usuario que ha intentado realizar un uso fraudulento de la aplicación.
- **attempts (intentos):** Número de intentos fraudulentos realizados por el usuario.

Tabla `ticket_prices`

- **type (tipo de tique):** Permite identificar el tipo de tique.
- **transport (tipo de transporte):** Permite identificar el tipo de transporte.
- **price (precio):** Precio de el tique correspondiente a un tipo de tique y un tipo de transporte.

La tabla `fraudulent_attempts` lleva un contador de los intentos de uso fraudulento que permite al sistema bloquear a un usuario que ha alcanzado el número máximo de intentos de uso fraudulentos. Para incluir automáticamente al usuario en la tabla `blacklist` una vez alcanzado dicho número se ha incluido un disparador en la base de datos. El número máximo de intentos fraudulentos es 3. El disparador se muestra a continuación:

```
CREATE OR REPLACE FUNCTION insert_blacklist() RETURNS
       TRIGGER AS $attempts_trigger$
DECLARE
BEGIN
       if(NEW.attempts=3) then
              INSERT INTO blacklist (imei, date, cause)
                     VALUES (OLD.imei, now(), 'FRAUDULENT');
              UPDATE fraudulent_attempts SET attempts=0
                     WHERE imei=OLD.imei;
       end if;
       RETURN NULL;
END;
$attempts_trigger$ LANGUAGE plpgsql;

CREATE TRIGGER attempts_trigger AFTER UPDATE
    ON fraudulent_attempts FOR EACH ROW
    EXECUTE PROCEDURE insert_blacklist();
```

Para poder tanto consultar como modificar la base de datos desde los servicios web implementados en Java se ha utilizado la JDBC de PostgreSQL. JDBC es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice.

6.3.5. Registro de eventos en logs

El sistema registra todos los eventos relacionados con la comunicación de los clientes con el servidor a través de los servicios web en logs que se almacenan en el mismo. Existen cuatro tipo de logs:

- **prices:** Log en el que se almacenan todos los eventos relativos a la consulta de precios de los tiques.
- **attempts:** Log en el que se almacenan todos los eventos relativos a la actualización de intentos de uso fraudulento llevados a cabo por los usuarios.
- **blacklist:** Log en el que se almacenan todos los eventos relativos a la consulta o modificación de la lista negra del sistema.
- **tickets:** Log en el que se almacenan todos los eventos relativos a la consulta, inserción o modificación de los tiques.

Los logs se nombran de la siguiente manera:

`[tipo de log]_log_[fecha del log].log`

7

Diseño de la aplicación de lectura NFC

La aplicación del lector consta de dos partes. Por un lado, la parte de lectura del tique enviado por el usuario que se implementa haciendo uso del modulo `nfcpy` escrito en Python. Por otro lado, la parte de validación del tique recibido que se implementa en Java y se comunica con la parte de lectura haciendo uso de la librería `Py4J`. Esto se explicará con más detalle en apartados posteriores.

En las siguientes secciones se describirá el diseño y la implementación de la aplicación del lector que recibe el tique enviado desde el teléfono móvil del usuario. La figura 7.1 muestra el diagrama de clases de la parte de la aplicación implementada en Java con los métodos más relevantes.

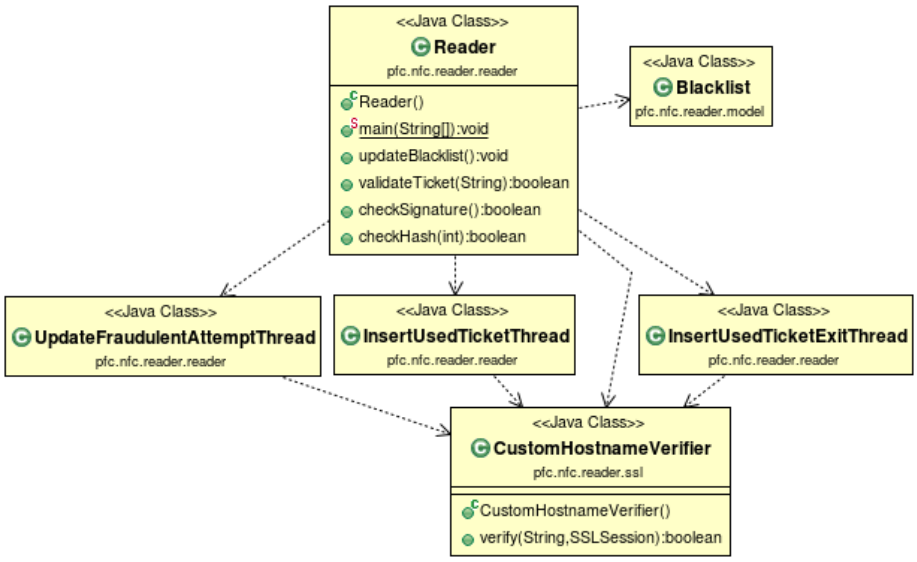

Figura 7.1: Diagrama de clases de la aplicación Java del lector

7.1. Definición de las clases

En el siguiente apartado se va a explicar la función de cada clase que aparece en el diagrama comentando los atributos y métodos más relevantes.

Clase `Reader`

Clase principal de la aplicación que muestra implementa los métodos que permite realizar la verificación de los tiques recibidos. Los métodos más destacados son:

- `updateBlacklist`: Este método se ejecuta periódicamente para actualizar la lista negra que se mantiene localmente en el equipo que ejecuta la aplicación lector. Se comunica con el servicio web que devuelve un listado con todos los imeis que incluidos en la lista negra del sistema.
- `validateTicket`: Este método identifica el tipo de tique que recibe (entrar, salir o revisor) y realiza la verificación correspondiente del tique en función de esto.
- `checkSignature`: Este método permite verificar la firma del tique enviado.
- `checkHash`: Este método permite verificar el código hash del tique recibido.

Clase `UpdateFraudulentAttemtpThread`

Clase que implementa un hilo que actualiza el número de intentos de uso fraudulento llevados a cabo por el usuario en la base de datos del servidor.

Clase `InsertUsedTicketThread`

Clase que implementa un hilo que actualiza la fecha de uso de un tique una vez utilizado por un usuario en la base de datos del servidor.

Clase `InsertUsedTicketExitThread`

Clase que implementa un hilo que actualiza el campo que permite conocer si un tique ha sido usado para salir de un servicio de transporte público en la base de datos del servidor.

Clase `CustomHostnameVerifier`

Clase que permite realizar la comunicación con los servicios web del servidor a través de HTTPS.

Clase `Blacklist`

Esta clase es una representación de un objeto que pueden ser serializado fácilmente en un objeto JSON que formara la respuesta que devolverá el servidor al consumidor del servicios web que permite actualizar la lista negra.

7.2. Implementación

7.2.1. Modulos y librerías

Como ya se comento en secciones anteriores, la aplicación del lector consta de dos partes. Por un lado, la parte de lectura del tique enviado por el usuario que se implementa haciendo uso del modulo `nfcpy` escrito en Python. Por otro lado, la parte de validación del tique recibido que se implementa en Java y se comunica con la parte de lectura haciendo uso de la librería `Py4J`.

El diseño se ha realizado de esta manera para aprovechar el código sobre la comprobación de la firma del tique ya realizado en Java en la parte de los servicios web del servidor y evitar así su reimplementación en Python.

El uso de una librería implementada en Python se debe principalmente a que de las tres opciones posibles para realizar la aplicación de lectura (`nfcpy`, `libnfc` y `nfctools`), `nfcpy` era la única que implementaba tanto el protocolo LLCP y el protocolo SNEP, era compatible con el lector empleado y además permitía la comunicación P2P entre el lector y un teléfono móvil Android.

Modulo `nfcpy`

El módulo `nfcpy` implementa las especificaciones del NFC Forum para el intercambio de datos inalámbrico de corto alcance con dispositivos y etiquetas NFC. Esta escrito en el lenguaje de programación Python y tiene como objetivo proporcionar una herramienta fácil de usar a la par que potente para aplicaciones Python.

Para poder funcionar correctamente necesita que el equipo donde se vaya a ejecutar tenga instalado Python 2.6 o superior pero no Python 3.x y la librería `pyUSB` que permite acceso a los puertos USB desde Python.

Actualmente el modulo cuenta con soporte para los siguientes lectores entre los que se encuentra el lector utilizado:

- Sony RC-S330/360/370
- **SCM SCL3711**
- ACS ACR122U (con limitaciones)

Para implementar la parte de lectura del tique enviado a través de NFC desde el teléfono móvil del usuario se han utilizado tres modulos de `nfcpy`:

- **Modulo** `nfc`: Modulo que permite el acceso al lector NFC.
- **Modulo** `nfc.ndef`: Modulo que permite codificar y decodificar mensajes y registros NDEF.
- **Modulo** `nfc.llcp`: Modulo que implementa el protocolo LLCP.
- **Modulo** `nfc.snep`: Modulo que implementa un servidor y un cliente para el protocolo SNEP definido por el NFC Forum.

`Py4J`

Py4J permite a programas Python que se ejecutan en un interprete de Python acceder dinámicamente a objetos Java en un Máquina Vitual Java. Los métodos se llaman como si los objetos Java residieran en el intérprete de Python.

Por defecto, el programa en Java escuchará por el puerto 25333 y permanecerá a la espera de recibir peticiones desde un programa en Python. El programa en Python llamará al método que permite validar los tiques enviados y recibirá como respuesta un booleano (True o False) que le indicará si el tique enviado es válido o no.

A continuación se muestra la parte de código que hay que incluir en la aplicación Java para que Py4J funcione correctamente. Básicamente se ejecuta un objeto `Reader` que implementa los métodos necesarios en el `GatewayServer` establecido por Py4J.

```
public static void main(String[] args) {
    Reader app = new Reader();
    // app is now the gateway.entry_point
    GatewayServer server = new GatewayServer(app);
    server.start();
  }
```

En la parte de Python se crea un `JavaGateway` que permite acceder al objeto que se ejecuta en la Máquina Virtual de Java, se coge la instancia del objeto y se llama al método.

```
# connect to the JVM
gateway = JavaGateway()
# get the Reader instance
check_signature_app = gateway.entry_point
# call the checkSignature method
if check_signature_app.checkSignature(ndef_message.data):
        log.info("<<<<<Ticket correcto>>>>>")
else:
        log.info("<<<<<Ticket incorrecto>>>>>")
```

7.2.2. Actualización periódica de la lista negra

Como ya se explicó en secciones anteriores, la validación del tique debe ser *offline*. Esto es que la aplicación del lector no debe comunicarse con el servidor para verificar la autenticidad y validez del tique.

Para poder comprobar localmente si el usuario que envía el tique está en la lista negra del sistema se ha implementado un método que actualiza periódicamente la lista negra y se almacena en la memoria volátil del equipo que ejecuta la aplicación del lector. Antes de proceder a la verificación del tique se comprueba que el usuario no este en la lista negra. Si no está se procede a verificar el tique pero si el usuario aparece en la lista negra no se le permite el acceso al servicio de transporte público.

Si el tique enviado fuera falso, la firma del mismo sería errónea por lo que el sistema no permitiría el acceso y el sistema incrementaría el número de intentos fraudulentos por parte del usuario.

Por otro lado, se toma como base que el sistema operativo Android no permite el acceso de otras aplicaciones a la información almacenada por la aplicación si no tiene permiso por lo que los tiques no podrán ser modificados una vez almacenados en el dispositivo.

Parte IV

Conclusiones y Presupuesto

8

Conclusiones

En este documento se ha descrito un sistema que permite adquirir y usar tiques de transporte público a través de un teléfono móvil. El sistema propuesto pretende complementar al sistema tradicional agilizando el proceso de compra y facilitando al usuario el almacenamiento y uso de los tiques. Sin embargo, en la actualidad no puede sustituir al sistema tradicional principalmente por dos motivos: en primer lugar, hoy en día no todos los teléfonos móviles disponen de chip NFC y en segundo lugar, no todos los usuarios de transporte público disponen de un *smartphone*.

A continuación, se comentarán posibles lineas de trabajo futuro que permitirán ampliar el sistema y las conclusiones que se han sacado durante el desarrollo e implementación del mismo.

8.1. Trabajo futuro

En este apartado se explican futuras líneas de trabajo del sistema que pueden abarcarse con objetivo de que se pueda continuar en un futuro mejorando la funcionalidad explicada, destacando una serie de aspectos a mejorar con objeto de una futura implementación.

8.1.1. Inclusión de un sistema de pago

El prototipo implementado no incluye un sistema de pago de tiques por lo que sería conveniente incluirlo en la aplicación. Se recomienda implementar el pago a través de Paypal principalmente por estar muy extendido entre los usuarios de Internet y la comodidad que ofrece al poder pagar directamente desde una cuenta Paypal o mediante una tarjeta de débito/crédito.

PayPal es una empresa estadounidense, propiedad de eBay, perteneciente al sector del comercio electrónico por Internet que permite la transferencia de dinero entre usuarios que tengan correo electrónico, una alternativa al tradicional método en papel como los cheques o giros postales. PayPal también procesa peticiones de pago en comercio electrónico y otros servicios webs, por los que cobra un porcentaje al vendedor. La mayor parte de su clientela proviene del sitio de subastas en línea eBay.

Paypal acaba de presentar un SDK para Android que permite realizar las compras «in-app» (sin cambiar de aplicación para realizar el pago) que facilita esta tarea. Además, en caso de utilizar una tarjeta de débito/crédito para pagar no será necesario introducir los datos, basta con tomar una fotografía de la tarjeta y utilizando la tecnología *card.io* se extraerán de manera segura los datos de la misma que permitirán realizar la compra.

8.1.2. Cálculo de precio de tiques por ruta

Otra de las optimizaciones que pueden aplicarse al sistema es la creación de un servicio web que permita calcular la tarifa de un tique en función de las estaciones de inicio y fin de trayecto. Para ello habría que modificar la aplicación añadiendo un par de «spinners» que permitan al usuario seleccionar las estaciones de inicio y fin de trayecto en función del tipo de transporte seleccionado en caso de que el billete sea de tipo sencillo. Si esta funcionalidad se implementara el formato de los billetes cambiaría para incluir la estación de entrada y la estación de salida del servicio de transporte público.

En el caso de un billete de autobús el usuario podrá seleccionar las paradas de inicio y fin en función de la línea de autobuses para la que vaya a adquirir el tique.

8.1.3. Nuevos tipos de tique

El sistema es tanto ampliable a otro tipo tanto de tique como de transporte como reducible. Pueden incluirse otro tipo de billetes como el bono de 10 viajes o el abono anual. También puede incluirse la posibilidad de adquirir el abono tanto mensual como anual por zonas (tal y como se comercializa en Madrid). Si esta funcionalidad se implementara el formato de los billetes cambiaría para incluir las zonas válidas para dicho abono.

Tomando como ejemplo España, no todas las provincias/ciudades cuentan con los mismos servicios de transporte público. El autobús es el medio de transporte público más común pero no todas las provincias/ciudades cuentan con red de Cernanías o red de Metro.

En otros casos, existen provincias que además de los medios de transporte incluidos también cuentan con una red de transporte marítimo como sucede en Cádiz que cuenta con dos líneas de servicio marítimo (B-042 Cádiz – El Puerto de Santa María y B-065 Cádiz – Rota). El sistema permite también la inclusión de este tipo de tiques.

8.1.4. Extensibilidad del sistema a otro tipo de tiques

La aplicación en un principio está pensada para ser utilizada en el ámbito del transporte público pero su uso puede extenderse a otros servicios que empleen un tique o billete para permitir el acceso como pueden ser:

- Entradas de teatro.
- Entradas para un concierto.
- Acceso y salida de un aparcamiento.

- Billetes de avión.
- Billetes de tren.
- ...

8.1.5. Redes sociales

Con el auge de las redes sociales como Twitter, Facebook o Foursquare podría implementarse un módulo que permitiera al usuario publicar una entrada en la red social cuando utilizase un tique tanto para acceder como para salir en el caso de las dos primeras o hacer *check-in* en el lugar (en este caso estación o parada) donde se emplea un tique en el caso de las dos últimas. Esta opción sería configurable de manera que la decisión final de publicar o no la entrada recaiga en el usuario.

8.2. Conclusiones personales

Durante el desarrollo del proyecto se han adquirido una gran cantidad de conocimientos ya que ha sido una gran labor de integración de distintas tecnologías entre las que se incluyen comunicaciones de red, utilización de bases de datos, nociones de seguridad, programación para dispositivos móviles Android, diseño de interfaces de usuario y estudio de la tecnología NFC.

El proyecto, además de abarcar nuevos temas, cubre prácticamente la totalidad de las materias impartidas en la titulación con la excepción de las de inteligencia artificial. Esto ha permitido completar mi formación principalmente en el ámbito de una de mis especialidades como es aplicaciones y sistemas distribuidos descubriendo la utilización práctica de muchos conocimientos adquiridos durante la carrera y sirviendo de complemento a las prácticas de las asignaturas de dicha especialidad.

A continuación se detallan los puntos que se han considerado más relevantes durante el desarrollo del proyecto:

- **Estudio de la tecnología NFC:** Antes de comenzar la implementación del proyecto se llevó a cabo un estudio de la tecnología NFC sobre la que no se tenían conocimientos previos. Este estudio permitió comprender qué es la tecnología NFC, cómo funciona, las distintas formas de comunicación disponibles y el formato de los mensajes intercambiados durante la comunicación. Durante este estudio también se investigó la forma en la que Android permitía enviar los mensajes NDEF y se buscó una librería que permitiera la comunicación P2P a través del protocolo que usa Android.
- **Desarrollo de aplicaciones para dispositivos Android:** Con motivo del creciente auge de los dispositivos móviles, se ha encontrado muy interesante el aprendizaje sobre como programar aplicaciones para dispositivos Android. Se estudió el funcionamiento del sistema operativo y se consulto la API disponible en la página de desarrolladores para aprender las bases sobre cómo programar aplicaciones Android. Los conocimientos sobre Android adquiridos durante el desarrollo de este proyecto han sido de gran ayuda de cara a futuros desarrollos de aplicaciones para la plataforma.

- **Implementación de servicios web RESTful:** En algunas materias de la carrera se trataron los servicios web explicando las diferencias entre un servicio web SOAP y un servicio web RESTful e incluso se llegó a a programar un servicio web SOAP muy simple en C#. El trabajo realizado durante el desarrollo del proyecto ha permitido consolidar estos conocimientos y aplicarlos de manera práctica viendo las distintas posibilidades de implementación ofrecidas por distintas tecnologías.

- **Uso de bases de datos PostgreSQL:** El diseño del sistema ha permitido una toma de contacto con motores de bases de datos que no se conocían previamente como ha sido PostgreSQL. Se ha considerado de gran utilidad el estudio de las ventajas e inconvenientes que ofrecían las distintas posibilidades con las que se podía implementar el sistema.

- **Refuerzo de cocimientos de seguridad:** El desarrollo del proyecto ha permitido consolidar conocimientos vistos durante la carrera sobre seguridad tanto en comunicaciones (HTTPS) como en mecanismos de protección de datos, control de integridad o autenticación. En el proyecto se emplean técnicas ya conocidas como firma digital, función resumen o cifrado, y otras que se han aprendido durante la fase de análisis como las funciones de derivación de clave.

Además de consolidar conocimientos adquiridos a lo largo de la carrera, he obtenido otros nuevos que creo me han hecho mejorar como ingeniera principalmente ampliando el rango de disciplinas dentro de la informática sobre el que tengo conocimiento, lo que me ayudará a mejorar los proyectos futuros de manera que pueda aplicar la tecnología idónea para resolver cada problema.

9

Presupuesto

En este capítulo se calculará el presupuesto del proyecto, incluyendo el desglose en distintos tipos de gastos, tanto económicos como de tiempo. Todos los cálculos se efectúan con sus valores antes de impuestos y se expresan en euros.

9.1. Gastos directos

En primer lugar se consideran los gastos directos, es decir, aquellos que imputables directamente al proyecto. Esta categoría engloba los gastos de personal, la amortización de equipos y el material fungible utilizado durante el desarrollo del proyecto.

9.1.1. Gastos de personal

Al tratarse de un proyecto informático, el gasto directo proviene mayoritariamente del coste del personal. Para realizar este calculo, se estima el tiempo ocupado por cada fase del proyecto, representado en el diagrama de Gantt de la figura 9.1. Los tiempos están calculados para una sola persona con una dedicación parcial de alrededor de 30 horas semanales, lo que supondría el 75 % de una jornada completa.

Se ha tomado el salario medio correspondiente a un analista-programador según Infojobs Trends [19], que es de unos 27.000€ brutos anuales. El coste final se calcula multiplicando el coste anual por la duración del proyecto (28 semanas) y la dedicación del personal al proyecto (75 %):

$$C_{personal} = 27000€ \frac{28}{52} 0{,}75 = 10903{,}85€$$

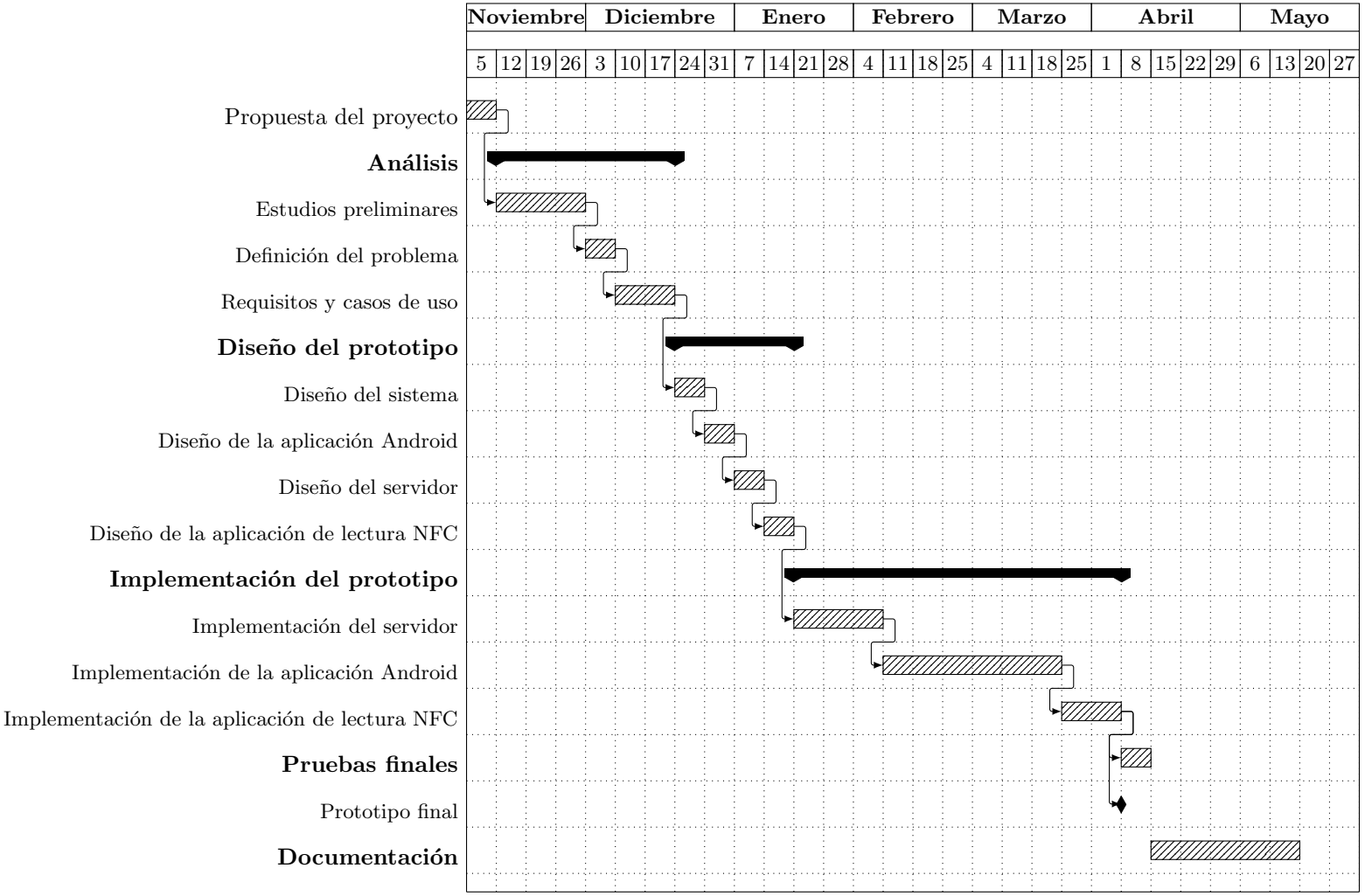

Figura 9.1: Diagrama de Gantt del proyecto

9.1.2. Gastos de hardware

Para desarrollar el proyecto han sido necesarios dos equipos (un servidor y un equipo de desarrollo) y un teléfono móvil Android con NFC. El servidor tiene instalado el motor de bases de datos y despliega los servicios web. En el equipo de desarrollo se ha implementado la aplicación Android y se ejecuta en una máquina virtual la aplicación del lector NFC. Se van a describir las características de los equipos y el teléfono empleado para posteriormente detallar el coste asociado a cada uno de ellos:

- **Equipo de desarrollo**
  - **Procesador:** Intel® Core™ 2 Quad Q6600 @ 2.4 GHz
  - **Memoria RAM:** 8 GB
  - **Disco duro:** 750 GB ATA
  - **Tarjeta de vídeo:** ATI Radeon HD 2400
- **Servidor**
  - **Procesador:** Intel® Core™ 2 Quad Q6600 @ 2.4 GHz
  - **Memoria RAM:** 8 GB
  - **Disco duro:** 750 GB ATA
  - **Tarjeta de vídeo:** ATI Radeon HD 2400
- **Samsung Galaxy Nexus**
  - **Procesador:** TI OMAP 4460 dual-core 1.2 GHz
  - **Memoria RAM:** 1 GB
  - **Almacenamiento interno:** 16 GB
  - **Tarjeta de gráfica:** GPU PowerVR SGX540

Equipo	Coste	Dedicación	Amortización	Coste imputable
Equipo de desarrollo	655.18€	28 semanas	36 meses	88.20€
Servidor	655.18€	3 semanas	36 meses	9.45€
Teléfono Móvil	328.00€	6 semanas	36 meses	9.46€
Lector NFC SCL3711	31.6€	2 semanas	36 meses	0.30€
TOTAL				**107.41€**

Tabla 9.1: Gastos de hardware

El periodo de amortización de los equipos empleados es de 3 años por lo que el coste imputable al proyecto, teniendo en cuenta que se utiliza en varios proyectos con dedicación igual a la del personal, la fórmula de cálculo del coste imputable al proyecto será:

$$C_{hardware} = \text{coste sin IVA} \frac{\text{semanas dedicadas al proyecto}}{\text{semanas lectivas anuales} * \text{periodo de amortización}} 0{,}75$$

9.1.3. Gastos de software

El equipo de desarrollo en el que se han implementado tanto la aplicación móvil *Android* como la aplicación de lectura NFC usa un sistema operativo *Windows 7 Professional* adquirido con una licencia gratuita para estudiantes a través de la universidad. Para realizar los diagramas de despliegue y los esquemas de las bases de datos se utilizó el programa de pago *Altova UModel Basic Edition*. El resto del software empleado no ha supuesto costes para el proyecto.

La documentación del proyecto ha sido realizada con el sistema de composición de textos LaTeXy las librerías empleadas tienen licencia libre así como el motor de la base de datos, el sistema operativo instalado en el servidor (*Ubuntu 10.04 LTS*), el entorno de desarrollo (*Eclipse*) y el software usado para montar la máquina virtual (*VMWare Player*).

Software	**Coste**
Licencia de Microsoft Windows Professional 7	0.00€
Licencia de Ubuntu 10.04 LTS	0.00€
Licencia de LaTeX	0.00€
Licencia de VMWare Player	0.00€
Licencia de PostgreSQL	0.00€
Licencia de Altova UModel Basic Edition	119.00€
Licencia de Android SDK	0.00€
TOTAL	**119.00€**

Tabla 9.2: Gastos de software

9.1.4. Gastos de material fungible

La mayor parte de los recursos consultados han sido en línea por lo que el único coste imputable en esta categoría es la compra de un lote de material de papelería (papel, tinta de impresora, CDs, bolígrafos etc.) por 35€.

9.2. Gastos indirectos, riesgo y beneficio

También se incluyen en el presupuesto los costes indirectos asociados al proyecto, lo que incluye gastos que son difíciles de imputar directamente al proyecto como pueden ser el mantenimiento del local o el coste de la electricidad. Para ello se estima un porcentaje del 20 % sobre los gastos directos.

Por último, se añade un porcentaje de riesgo cuya finalidad es sufragar posibles errores en el presupuesto. Dado que es un proyecto de corta duración, un retraso de una semana representa aproximadamente un 5 % de error. Para tener un margen de error de al menos dos semanas, se utilizará como riesgo el 10 % de los costes totales (directos e indirectos) del proyecto.

Hay que tener en cuenta que el principal objetivo del desarrollo del proyecto es ganar dinero, por lo que se suma al coste total del proyecto el margen de beneficio deseado. En este caso se establece al 10 % de los gastos del proyecto.

9.3. Tabla resumen y totales

En la tabla 9.3 se resumen y totalizan todos los gastos del proyecto detallados en las secciones previas.

Concepto	Base	Porcentaje	Total
Total gastos directos			**11165,26**
Gastos de personal			10903,85
Gastos de hardware			107,41
Gastos de software			119,00
Material fungible			35,00
Gastos indirectos	11165,26	20 %	2233,05
Total gastos			**13398,31**
Riesgo	13398,31	10 %	1339,83
Beneficio	13398,31	10 %	1339,83
Total sin IVA			**16077,97**
IVA	16077,97	21 %	3376,37
Total			**19454,34**

Tabla 9.3: Tabla resumen del presupuesto

El coste total del proyecto es de dieciséis mil setenta y siete euros y noventa y siete céntimos (16077,97€) sin incluir el impuesto sobre el valor añadido. Aplicando el IVA vigente a Mayo de 2013 (21 %) el total es de diecinueve mil cuatrocientos cincuenta y cuatro euros y treinta y cuatro céntimos (19454,34€), aunque siempre se aplicará el IVA vigente en el momento del pago del proyecto.

Glosario

- **3G:** Abreviación de tercera generación de transmisión de voz y datos a través de telefonía móvil mediante UMTS (Universal Mobile Telecommunications System o servicio universal de telecomunicaciones móviles). Los servicios asociados con la tercera generación proporcionan la posibilidad de transferir tanto voz y datos (una llamada telefónica o una videollamada) y datos no-voz (como la descarga de programas, intercambio de correos electrónicos, y mensajería instantánea).
- **API (del inglés *Application Programming Interface*):** Interfaz de programación de aplicaciones. Conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
- **App:** Aplicación software diseñada para ser ejecutada en *smartphones*, *tablets* o otros dispositivos móviles.
- **Bytecode:** Código intermedio más abstracto que el código máquina. Habitualmente es tratado como un archivo binario que contiene un programa ejecutable similar a un módulo objeto, que es un archivo binario producido por el compilador cuyo contenido es el código objeto o código máquina .
- **C#:** Lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270). Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.
- **CPU (del inglés *Central Processing Unit*):** Unidad Central de Procesamiento, también llamado microprocesador o simplemente procesador, es el componente principal del ordenador y otros dispositivos programables, que interpreta las instrucciones contenidas en los programas y procesa los datos.
- **Eclipse:** Programa informático compuesto por un conjunto de herramientas de programación. de código abierto multiplataforma para desarrollar lo que el proyecto llama .ªplicaciones de Cliente Enriquecido". Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

- **ESA (del inglés European Space Agency):** La Agencia Espacial Europea es una organización intergubernamental dedicada a la exploración espacial. Con 18 estados miembros, fue constituida el 31 de mayo de 1975. Emplea a unas 2000 personas (excluyendo subcontratados) y tiene un presupuesto anual en torno a los 3.600 millones de euros.
- **Facebook:** Empresa creada por Mark Zuckerberg y fundada junto a Eduardo Saverin, Chris Hughes y Dustin Moskovitz consistente en un sitio web de redes sociales. Originalmente era un sitio para estudiantes de la Universidad de Harvard, pero actualmente está abierto a cualquier persona que tenga una cuenta de correo electrónico.
- **Google Play (antes Android Market):** Tienda de software en línea desarrollada por Google para los dispositivos con sistema operativo Android. Es una aplicación que está preinstalada en la mayoría de los dispositivos Android y que permite a los usuarios buscar, obtener información y descargar aplicaciones publicadas por desarrolladores terceros.
- **GPS (del inglés *Global Positioning System*):** sistema global de navegación por satélite (GNSS) que permite determinar en todo el mundo la posición de un objeto, una persona o un vehículo con una precisión hasta de centímetros (si se utiliza GPS diferencial), aunque lo habitual son unos pocos metros de precisión. El sistema fue desarrollado, instalado y actualmente operado por el Departamento de Defensa de los Estados Unidos.
- **GPU (del inglés *Graphics Processing Unit*):** La unidad de procesamiento gráfico es un coprocesador dedicado al procesamiento de gráficos u operaciones de coma flotante, para aligerar la carga de trabajo del procesador central en aplicaciones como los videojuegos y o aplicaciones 3D interactivas. De esta forma, mientras gran parte de lo relacionado con los gráficos se procesa en la GPU, la unidad central de procesamiento (CPU) puede dedicarse a otro tipo de cálculos (como la inteligencia artificial o los cálculos mecánicos en el caso de los videojuegos).
- **GUI (del inglés *Graphical User Interface*):** La interfaz gráfica de usuario es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador.
- **HTTPS (del inglés *Hypertext Transfer Protocol Secure*):** Protocolo seguro de transferencia de hipertexto es un protocolo de aplicación basado en el protocolo HTTP, destinado a la transferencia segura de datos de Hipertexto, es decir, es la versión segura de HTTP. Es utilizado principalmente por entidades bancarias, tiendas en línea, y cualquier tipo de servicio que requiera el envío de datos personales o contraseñas.
- **IMEI (del inglés *International Mobile Equipment Identity*):** Identidad Internacional de Equipo Móvil es un código pre-grabado en los teléfonos móviles GSM. Este código identifica al aparato unívocamente a nivel mundial, y es transmitido por el aparato a la red al conectarse a ésta.

- **Java:** Lenguaje de programación originalmente desarrollado por James Gosling de Sun Microsystems (la cual fue adquirida por la compañía Oracle) y publicado en el 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. El lenguaje deriva mucho de su sintaxis de C y C++, pero tiene menos facilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java) que puede correr en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora.
- **JDBC (del inglés *Java Database Connectivity*):** API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice.
- **JSON (del inglés *JavaScript Object Notation*):** Formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.
- JSP (del inglés Java Server Pages): Las Java Server Pages es un método de creación de páginas web dinámicas en servidor usando el lenguaje Java. Se ejecutan en una máquina virtual Java, lo cual permite que, en principio, se puedan usar en cualquier tipo de ordenador, siempre que exista una máquina virtual Java para él.
- **JVM (en inglés *Java Virtual Machine*):** Una máquina virtual Java es una máquina virtual de proceso nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el bytecode Java), el cual es generado por el compilador del lenguaje Java.
- **Kernel:** Software que constituye la parte más importante del sistema operativo. Es el principal responsable de facilitar a los distintos programas acceso seguro al hardware de la computadora o en forma básica, es el encargado de gestionar recursos, a través de servicios de llamada al sistema.
- **Licencia BSD (del inglés *Berkeley Software Distribution*):** Es una licencia de software libre permisiva como la licencia de OpenSSL o la MIT License. Esta licencia tiene menos restricciones en comparación con otras como la GPL estando muy cercana al dominio público. La licencia BSD al contrario que la GPL permite el uso del código fuente en software no libre.
- **Licencia GNU GPL (del inglés *GNU General Public License*):** Licencia más ampliamente usada en el mundo del software y garantiza a los usuarios finales (personas, organizaciones, compañías) la libertad de usar, estudiar, compartir (copiar) y modificar el software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios. Esta licencia fue creada originalmente por Richard Stallman fundador de la Free Software Foundation (FSF) para el proyecto GNU (GNU project).

- **Log:** Registro oficial de eventos durante un rango de tiempo en particular.
- **Máquina Virtual:** Software que simula a una computadora y puede ejecutar programas como si fuese una computadora real.
- **P2P (*Peer-to-peer*):** Red de computadoras en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí. Es decir, actúan simultáneamente como clientes y servidores respecto a los demás nodos de la red. Las redes P2P permiten el intercambio directo de información, en cualquier formato, entre los ordenadores interconectados.
- **PayPal:** empresa estadounidense, propiedad de eBay, perteneciente al sector del comercio electrónico por Internet que permite la transferencia de dinero entre usuarios que tengan correo electrónico, una alternativa al tradicional método en papel como los cheques o giros postales. PayPal también procesa peticiones de pago en comercio electrónico y otros servicios webs, por los que cobra un porcentaje al vendedor. La mayor parte de su clientela proviene del sitio de subastas en línea eBay.
- **POJO (del inglés *Plain Old Java Object*):** Sigla creada por Martin Fowler, Rebecca Parsons y Josh MacKenzie en septiembre de 2000 y utilizada por programadores Java para enfatizar el uso de clases simples y que no dependen de un framework en especial.
- **RAM (del inglés *Random-Access Memory*):** Memoria de trabajo para el sistema operativo, los programas y la mayoría del software. Es allí donde se cargan todas las instrucciones que ejecutan el procesador y otras unidades de cómputo. Se denominan «de acceso aleatorio» porque se puede leer o escribir en una posición de memoria con un tiempo de espera igual para cualquier posición, no siendo necesario seguir un orden para acceder a la información de la manera más rápida posible.
- **REST (del inglés *Representational State Transfer*):** Técnica de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web. El término se originó en el año 2000, en una tesis doctoral sobre la web escrita por Roy Fielding, uno de los principales autores de la especificación del protocolo HTTP y ha pasado a ser ampliamente utilizado por la comunidad de desarrollo.
- **SDK (del inglés *Software Development Kit*):** Conjunto de herramientas de desarrollo de software que le permite al programador crear aplicaciones para un sistema concreto, por ejemplo ciertos paquetes de software, frameworks, plataformas de hardware, computadoras, videoconsolas, sistemas operativos, etc.
- **Servlet:** Objetos que corren dentro y fuera del contexto de un contenedor de servlets (ej: Tomcat) y extienden su funcionalidad. La palabra servlet deriva de otra anterior, applet, que se refería a pequeños programas que se ejecutan en el contexto de un navegador web. Por contraposición. El uso más común de los servlets es generar todas páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web.

- **SOAP (del inglés *Simple Object Access Protocol*):** Protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. Este protocolo deriva de un protocolo creado por David Winer en 1998, llamado XML-RPC. SOAP fue creado por Microsoft, IBM y otros y está actualmente bajo el auspicio de la W3C. Es uno de los protocolos utilizados en los servicios Web.

- **SSH (del inglés *Secure SHell*):** Protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red. Permite manejar por completo la computadora mediante un intérprete de comandos, y también puede redirigir el tráfico de X para poder ejecutar programas gráficos si tenemos un Servidor X (en sistemas Unix y Windows) corriendo. SSH permite copiar datos de forma segura (tanto ficheros sueltos como simular sesiones FTP cifradas), gestionar claves RSA para no escribir claves al conectar a los dispositivos y pasar los datos de cualquier otra aplicación por un canal seguro tunelizado mediante SSH.

- **SSL/TLS (del inglés *Secure Sockets Layer* y *Transport Layer Security*):** Protocolos criptográficos que proporcionan comunicaciones seguras por una red, comúnmente Internet. SSL proporciona autenticación y privacidad de la información entre extremos sobre Internet mediante el uso de criptografía. Habitualmente, sólo el servidor es autenticado (es decir, se garantiza su identidad) mientras que el cliente se mantiene sin autenticar.

- **Tag:** Etiqueta RFID. Las etiquetas RFID son la forma de empaquetado más habitual de los dispositivos RFID.

- **Twitter:** servicio de microblogging, con sede en San Francisco, California, con filiales en San Antonio Texas y Boston (Massachusetts) en Estados Unidos. La red permite enviar mensajes de texto plano de corta longitud, con un máximo de 140 caracteres, llamados tuits, que se muestran en la página principal del usuario. Los usuarios pueden suscribirse a los tuits de otros usuarios – a esto se le llama "seguirz a los usuarios abonados se les llama "seguidores.º "followers".

- **UML (del inglés *Unified Modeling Language*):** Lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano"del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados.

- **XML (del inglés *eXtensible Markup Language*):** Lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C). Deriva del lenguaje SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML) para estructurar documentos grandes.

Bibliografía

[1] Reserach2guidance. *The Enterprise Mobile App Market Status Report 2012*. URL: http://www.research2guidance.com/shop/index.php/enterprise-mobile-app-market-status-report (visitado 26-03-2013).

[2] ISO/EIC 14443. "Identification Cards - Contactless integrated circuit cards - Proximity cards". En: (2001). URL: http://www.iso.org.

[3] *FeliCa*. URL: http://www.sony.net/Products/felica/ (visitado 14-01-2013).

[4] *MIFARE*. URL: http://mifare.net/ (visitado 14-01-2013).

[5] ECMA INTERNATIONAL. "Near Field Communication Interface and Protocol (NFCIP-1)". En: (dic. de 2004). URL: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-340.pdf (visitado 17-01-2013).

[6] *NFC Forum*. URL: http://www.nfc-forum.org/home/ (visitado 08-01-2013).

[7] ISO/IEC 18092. "Information Technology – Telecommunications and information exchange between systems – Near Field Communication – Interface and Protocol (NFCIP-1)". En: (2004). URL: http://www.iso.org.

[8] *Amplitude-Shift Keying*. Ene. de 2013. URL: http://en.wikipedia.org/w/index.php?title=Amplitude-shift_keying&oldid=527190295 (visitado 08-01-2013).

[9] *Miller Encoding*. Ene. de 2013. URL: http://en.wikipedia.org/w/index.php?title=Delay_encoding&oldid=524246978 (visitado 08-01-2013).

[10] *Manchester Code*. Ene. de 2013. URL: http://en.wikipedia.org/w/index.php?title=Manchester_code&oldid=531140068 (visitado 08-01-2013).

[11] Ernst Haselsteiner y Klemens Breitfuß. "Security in Near Field Communication (NFC) Strengths and Weaknesses". En: *Semiconductors* 11.71 (2006), pág. 71. URL: http://ece.wpi.edu/~dchasaki/papers/Security%20in%20NFC.pdf.

[12] *INTECO - Seguridad, Observatorio, Artículos*. URL: http://www.inteco.es/Seguridad/Observatorio/Articulos/CdN_NFC (visitado 17-01-2013).

[13] *Open Handset Alliance*. URL: http://www.openhandsetalliance.com/ (visitado 08-01-2013).

[14] *Juniper Research*. http://www.juniperresearch.com. URL: http://www.juniperresearch.com/reports/mobile_ticketing_evolution (visitado 13-02-2013).

[15] Windsor House Transport for London. *What is Oyster?* URL: `http://www.tfl.gov.uk/tickets/14836.aspx` (visitado 20-01-2013).

[16] European Space Agency. *Guide to applying the ESA software engineering standards to small software projects*. 1996. URL: `http://emits.esa.int/emits-doc/e_support/Bssc962.pdf`.

[17] *OpenSSL: The Open Source toolkit for SSL/TLS*. http://www.openssl.org/. URL: `http://www.openssl.org/` (visitado 05-05-2013).

[18] *Bouncy Castle*. http://www.bouncycastle.org/. URL: `http://www.bouncycastle.org/` (visitado 05-05-2013).

[19] *InfoJobs Trends*. http://salarios.infojobs.net. URL: `http://salarios.infojobs.net/resultados.cfm?sueldo=analista+programador` (visitado 13-05-2013).